BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

IMAGES ARE BEST AVAILABLE COPY.

As rescanning documents *will not* correct images, please do not report the images to the Image Problem Mailbox.

(19) JAPANESE PATENT OFFICE

PATENT ABSTRACTS OF JAPAN

(11) Publication number: 09274560 A

(43) Date of publication of application: 21.10.97

(51) Int. Cl G06F 7/552
G06F 7/72
G06F 9/305
G09C 1/00

(21) Application number: 08110057

(22) Date of filing: 05.04.96

(71) Applicant: OKI MICRO DESIGN MIYAZAKI:KK
OKI ELECTRIC IND CO LTDKAWASAKI DENKI:GOUSHI

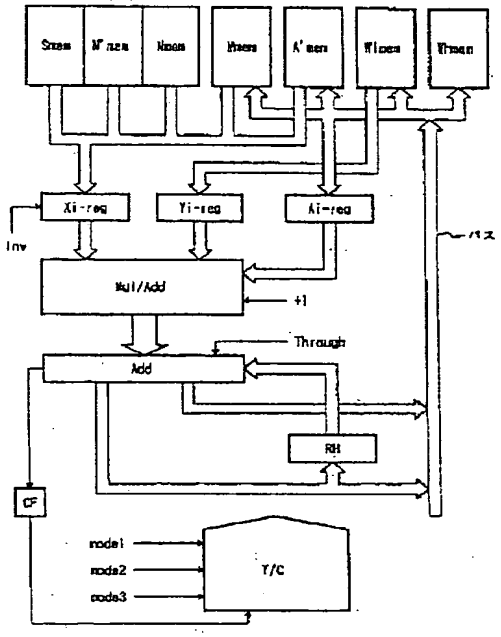
(72) Inventor: EBIHARA HIDENORI
KAWASAKI KIYOTO

(54) POWER REMAINDER OPERATION CIRCUIT, POWER REMAINDER OPERATION SYSTEM AND OPERATION METHOD FOR POWER REMAINDER OPERATION

(57) Abstract:

PROBLEM TO BE SOLVED: To execute various kinds of operations only by supplying a mode signal prepared in advance in the case of finding the solution of power remainder operation.

SOLUTION: At the power remainder operation circuit of M^e modN, this circuits is provided with an arithmetic part for executing the 1st operation of A.A.R' modN (R' is the inverse of R under the method of modN) in response to a timing control signal corresponding to a 1st mode signal while using an integer R which a prime with N and larger than N, executing the 2nd operation of A.B.R' modN in response to a timing control signal corresponding to a 2nd mode signal while using the integer R and executing the 3rd operation of A.1.R' modN in response to a timing control signal corresponding to a 3rd mode signal while using the integer R and a timing control circuit T/C for outputting the timing control signal corresponding to the 1st or 3rd mode signal to the arithmetic part.



(19)日本国特許庁（ＪＰ）　(12) **公開特許公報（Ａ）**　(11)特許出願公開番号

特開平9－274560

(43)公開日　平成９年(1997)10月21日

(51)Int.Cl.[6]		識別記号	庁内整理番号	ＦＩ			技術表示箇所
Ｇ０６Ｆ	7/552			Ｇ０６Ｆ	7/552	Ａ	
	7/72				7/72		
	9/305		7259－5Ｊ	Ｇ０９Ｃ	1/00	６５０Ａ	
Ｇ０９Ｃ	1/00	６５０		Ｇ０６Ｆ	9/30	３４０Ａ	

審査請求　未請求　請求項の数12　ＦＤ　（全 32 頁）

(21)出願番号　特願平8－110057

(22)出願日　平成８年(1996)４月５日

(71)出願人　591049893
株式会社沖マイクロデザイン宮崎
宮崎県宮崎市大和町９番２号

(71)出願人　000000295
沖電気工業株式会社
東京都港区虎ノ門１丁目７番12号

(71)出願人　596060365
合資会社川▲崎▼電機
宮崎県児湯郡高鍋町大字南高鍋569番地３

(72)発明者　海老原　秀徳
宮崎県宮崎市大和町９番２号　株式会社沖マイクロデザイン宮崎内

(74)代理人　弁理士　佐藤　幸男　（外１名）

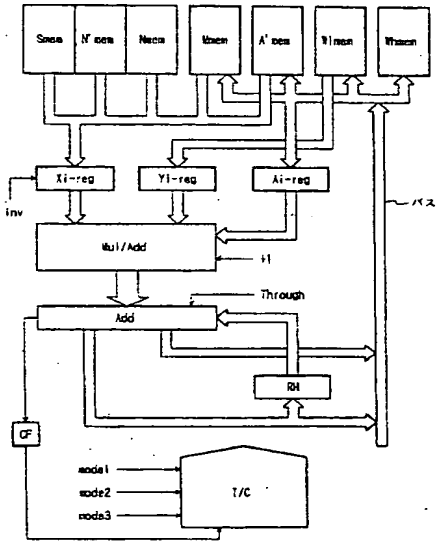
最終頁に続く

(54)【発明の名称】　べき乗剰余演算回路及びべき乗剰余演算システム及びべき乗剰余演算のための演算方法

(57)【要約】　（修正有）

【課題】　べき乗剰余演算の解を求めるに際し、予め用意されたモード信号を供給するのみで、各種演算を実行する。

【解決手段】　Ｍｅｍｏｄ Ｎの乗剰余演算回路において、Ｎと素であり、かつＮよりも大きい整数Ｒを用い、第１のモード信号に対応したタイミング制御信号に応答して、Ａ・Ａ・Ｒ’ ｍｏｄＮ（Ｒ’はｍｏｄ Ｎ法下でのＲのインバース）なる第１の演算を実行し、整数Ｒを用い、第２のモード信号に対応したタイミング制御信号に応答して、Ａ・Ｂ・Ｒ’ ｍｏｄＮなる第２の演算を実行し、整数Ｒを用い、第３のモード信号に対応したタイミング制御信号に応答して、Ａ・１・Ｒ’ ｍｏｄＮなる第３の演算を実行する演算部と、第１ないし第３のモード信号に対応したタイミング制御信号を演算部に出力するタイミング制御回路Ｔ／Ｃとを設ける。



第１の実施の形態

1

【特許請求の範囲】

【請求項１】　正の整数M、ｅ、Ｎに関するM^e mod Ｎなるべき乗剰余演算を行うべき乗剰余演算回路において、

Ｎと素であり、かつＮよりも大きい整数Ｒを用い、第１のモード信号に対応したタイミング制御信号に応答して、Ａ・Ａ・Ｒ’ ｍｏｄＮ（Ｒ’は、ｍｏｄＮ法下でのＲのインバースである。）なる第１の演算を実行し、前記整数Ｒを用い、第２のモード信号に対応したタイミング制御信号に応答して、Ａ・Ｂ・Ｒ’ ｍｏｄＮなる第２の演算を実行し、前記整数Ｒを用い、第３のモード信号に対応したタイミング制御信号に応答して、Ａ・１・Ｒ’ ｍｏｄＮなる第３の演算を実行する演算部と、

前記第１ないし第３のモード信号を受信し、前記第１ないし第３のモード信号に対応した前記タイミング制御信号を前記演算部に出力するタイミング制御回路とを有することを特徴とするべき乗剰余演算回路。

【請求項２】　前記第１ないし第３の演算の結果の前記Ｒ未満の値が０であることを検出する検出回路を設けたことを特徴とする請求項１記載のべき乗剰余演算回路。

【請求項３】　前記Ａもしくは Ｂのビット長に応じて、前記タイミング制御回路を制御する演算ビット長選択回路を設けたことを特徴とする請求項１記載のべき乗剰余演算回路。

【請求項４】　前記演算部は、第４のモード信号に対応したタイミング制御信号に応答して、Ａ・Ｂ＋Ｃなる第４の演算を実行し、前記タイミング制御回路は、前記第４のモード信号を受信し、前記第４のモード信号に対応した前記タイミング制御信号を出力することを特徴とする請求項１記載のべき乗剰余演算回路。

【請求項５】　前記演算における乗算値を格納する乗算値格納部と、

前記演算における被乗算値を格納する複数個の被乗算値格納部と、

複数の前記被乗算値格納部のいずれかの出力を選択して前記演算部に出力する選択回路と、

前記複数個の被乗算値格納部に各々対応して設けられ、前記演算部の出力のうちの上位桁もしくは下位桁のいずれか一方を格納する複数個の演算結果格納部とを設けたことを特徴とする請求項１記載のべき乗剰余演算回路。

【請求項６】　請求項１記載のべき乗剰余演算回路と外部装置との間に設けられ、前記外部装置の命令に応じて前記演算部の動作を制御する演算コントロール回路を設けたことを特徴とするべき乗剰余演算システム。

【請求項７】　前記第１、第２もしくは第３の演算のいずれかが終了したことを検出して、前記外部装置に割り込み要求信号を出力する割り込み制御回路を設けたことを特徴とする請求項６記載のべき乗剰余演算システム。

【請求項８】　前記第１、第２もしくは第３の演算のいずれかが実行中であることを検出して、スリープ信号を出力するスリープ制御回路と、

前記スリープ信号を受信している間、クロック信号を前記外部装置に供給するクロック制御回路とを設けたことを特徴とする請求項６記載のべき乗剰余演算システム。

2

【請求項９】　第１のクロック信号を受信して、該第１のクロック信号を逓倍した第２のクロック信号を前記外部装置もしくは前記演算部に供給する逓倍速クロック制御回路を設けたことを特徴とする請求項６記載のべき乗剰余演算システム。

【請求項１０】　正の整数M、ｅ、Ｎに関するM^e mod Ｎなるべき乗剰余演算を、Ｎと素であり、かつＮよりも大きい整数Ｒを用いて実行するべき乗剰余演算の演算方法であって、乗算値を格納する乗算値格納部及び被乗算値を格納する被乗算値格納部とを有し、前記乗算値格納部に格納された乗算値と、前記被乗算値格納部に格納された被乗算値とを乗算した結果に対してＲ’ ｍｏｄＮ（Ｒ’は、ｍｏｄＮ法下でのＲのインバースである。）を実行する演算回路を使用して、前記べき乗剰余演算のための演算方法において、

前記乗算値格納部及び前記被乗算値格納部にＲｍｏｄＮを与えることにより第１の演算結果を得る第１の演算ステップと、

前記乗算値格納部に前記第１の演算結果を与え、前記被乗算値格納部にR^2ｍｏｄＮとMとを乗算した演算結果を与えることにより第２の演算結果を得る第２の演算ステップと、

前記乗算値格納部に前記第１の演算結果もしくは前記第２の演算結果を与え、前記被乗算値格納部に１を与えることにより第３の演算結果を得る第３の演算ステップとを有し、

前記第１ないし第３の各演算ステップにおける演算は、前記第１ないし第３の各演算結果をＸとして、

ｍ＝（ＸｍｏｄＲ）・Ｎ’ ｍｏｄＲなる第４の演算（Ｎ’は、ｍｏｄＮ法下でのＮのインバースである。）を実行する第４の演算ステップと、その後、

ｔ＝（Ｘ＋ｍ・Ｎ）／Ｒなる第５の演算を実行する第５の演算ステップとを含むことを特徴とするべき乗剰余演算のための演算方法。

【請求項１１】　前記第４の演算は、ＸのＲ未満の値とＮ’とを乗算し、この乗算結果のＲ未満の値をｍとするステップからなり、

前記第５の演算は、ＸのＲ未満の値が全て０であることを検出した時には、このＸのＲ以上の値をｔとし、ＸのＲ未満の値が少なくとも１を含むことを検出した時には、（ＸのＲ以上の値＋ｍ・ＮのＲ以上の値）＋１をｔとするステップからなることを特徴とする請求項１０記載の演算方法。

【請求項１２】　前記第５の演算ステップはさらに、Ｒ≦ｔの関係を検出した時はｔ－Ｎ実行することを特徴とする請求項１０記載の演算方法。

【発明の詳細な説明】

【０００１】

【産業上の利用分野】本発明は、情報通信ネットワーク、交通、金融、医療、流通等の分野において使用される情報の暗号化技術及び復号化技術に関するものであり、特にこれらの情報の暗号化及び復号化を実現するためのべき乗剰余演算回路及びそのシステム及びべき乗剰余演算のための演算方法に関するものである。

【０００２】

【従来の技術】情報通信技術の発展に伴い、情報ネットワーク上のセキュリティーの確保（データの盗用や破壊を防止する）が重要視されるようになってきている。そのために、情報の暗号化技術及び復号化技術が、情報通信分野にとどまらず、交通、金融、医療、流通等の身近な分野で使用されつつある。従って、この種の暗号化技術及び復号化技術は、高度なセキュリティーが単純な原理によって実現できることが要求される。まず、この種の技術の理解を容易にするため、情報の暗号化・復号化についての概略を説明する。

【０００３】暗号の世界においては、”非対象アルゴリズム”が質的に優れている。非対象暗号アルゴリズムとは、暗号化鍵と復号化鍵が異なっており、そのいずれか一方から他方が”容易に計算できない”暗号アルゴリズムをいう。

【０００４】この非対象暗号アルゴリズムの代表的なものに、べき乗剰余演算を用いるタイプのＲＳＡ暗号、エルガマル暗号、ラビン暗号、ウイリアムス暗号等がある。そして、暗号アルゴリズムを応用する上では、”ディジタル署名”のシステムがあり、現在その標準化の動きがある。その対象となっている代表的なものは、ＲＳＡ署名法、エルガマル署名法、シュノア署名法、ＤＳＡ（Digital Sigunature Algorithm）署名法等であり、これらは全て長いビット長のべき乗剰余演算を使用するタイプのものである。従って、これらのディジタル署名のシステムを実現する上では、長いビット長のべき乗剰余演算を短時間で終了することのできる演算器の開発が必要不可欠である。

【０００５】上で説明したＲＳＡ暗号、エルガマル暗号、ラビン暗号、ウイリアムス暗号等は、次式（１）のべき乗剰余演算の形式が基本として使用される。式（１）は、X^YをＮで割った時の余りを求めることを意味する。また、式（１）において、Ｘは暗号化（復号化）の対象となる平文、Ｙ及びＮは暗号化（復号化）のための鍵（キー）である。

【０００６】$X^Y \bmod N \cdots (1)$

このべき乗剰余演算を用いることにより、情報の暗号化及び復号化が容易に実行され、かつＸ、Ｙ、Ｎのオペランドビット長を長くすることで、各鍵の解読を困難にすることができる。

【０００７】しかし、オペランドビット長を長くすると、べき乗剰余演算に長時間を要することになる。そこで、オペランドビット長が長いべき乗剰余演算をいかに短時間に終了させるかがポイントとなる。

【０００８】さて、ＲＳＡ暗号を例にとり、べき乗剰余演算を使用した実際の暗号化・復号化、その使われ方を以下に説明する。

【０００９】（１）ＲＳＡ暗号の暗号化・復号化の概略

暗号化には、

$C = M^e \bmod n \cdots (2)$

なる式を使用する。

【００１０】復号化には、

$M = C^d \bmod n \cdots (3)$

なる式を使用する。ここでＭは、暗号化の対象となる平文、Ｃは暗号化された平文すなわち暗号文である。そして、式（２）におけるｅ、ｎは暗号化鍵、式（３）におけるｄ、ｎは復号化鍵であり、

$$n = p \cdot q \cdots (4)$$

$$1 \equiv e \cdot d \bmod \{LCM(p-1, q-1)\} \cdots (5)$$

れる。

（≡は、左辺と右辺が相似であること、ＬＣＭは、最小公倍数を意味する。またｐとｑとは互いに素な整数である。）なる条件があらかじめ与えられている。なお、ｅ、ｎは公開鍵、ｄ、ｐ、ｑは秘密鍵である。

【００１１】以上の式（４）（５）は、ともに暗号アルゴリズム上のべき乗剰余演算の数値の条件を定義しており、式（４）は、ｎは互いに素な大きな素数ｐとｑとの積であることを示している。今、ｐとｑはともに奇数なので、当然ｎは奇数でなければならない。次に式（５）は、式（４）で示したｐ及びｑの１だけ小さい値同士の最小公倍数でｅとｄとの積ｅ・ｄを割ったときの余りが１になることを示している。

【００１２】以上の式（４）（５）のような条件に基づき、平文Ｍは式（２）を用いて暗号化され、また暗号化された平文Ｍ（暗号文Ｃ）は式（３）を用いて復号化される。

【００１３】（２）暗号化・復号化の事例

次に具体的な事例として、「発信者Ａは平文Ｍを暗号文Ｃに暗号化して送信し、受信者Ｂは暗号文Ｃを平文Ｍに復号化する。」という伝達（ディジタル署名あり）を行った場合の、発信者Ａ及び受信者Ｂが行う処理方法を図２を用いて説明する。

【００１４】送信者Ａが行う処理

・自分が作成した平文ＭＡを自分の秘密鍵d_Aを使って変形し、署名文ＣＡを作成する。（署名）

$CA \equiv MA^{d_A} \bmod nA \cdots (6)$

・Ｂの公開鍵ｅＢを使って、暗号化署名文ｃＡを作成する。（暗号化）

$cA \equiv CA^{e_B} \bmod nB \cdots (7)$

・ｃＡをＢへ送信する。

5　　6

【００１５】受信者Ｂが行う処理
・受信した暗号化署名文ｃＡを自分の秘密鍵ｄＢを使っ＊て変形する。（復号化）

$$cA^{dB} \bmod nB \equiv (CA^{eB} \bmod nB)^{dB} \bmod nB \cdots (8)$$

ここで、$CA^{eB}=X$とおくと、式（８）は、

$$(CA^{eB} \bmod nB)^{dB} \bmod nB = (X \bmod nB)^{dB} \bmod nB \cdots (9)$$

と変形できる。

【００１６】ここで、式（９）中の$X \bmod nB=Y$とおく、すなわちＸをｎＢで割ったときの余りがＹでありそのときの商がｋとすると、

$$(X \bmod nB)^{dB} \bmod nB = Y^{dB} \bmod nB = (X-k \cdot nB)^{dB} \bmod nB \cdots (11)$$

となる。式（１１）の$(X-k \cdot nB)^{dB}$を展開する★　★と、定数ａｉ（ｉ＝１、２、・・・）を用いて、

$$(X-k \cdot nB)^{dB} = (X^{dB} - a1 \cdot X^{dB-1} \cdot nB + a2 \cdot X^{dB-2} \cdot nB^2 - \cdots - ai \cdot nB^{dB}) \cdots (12)$$

と表すことができる。この式（１２）を式（１１）に代☆　☆入すると、

$$(X \bmod nB)^{dB} \bmod nB = Y^{dB} \bmod nB = (X-k \cdot nB)^{dB} \bmod nB = (X^{dB} - a1 \cdot X^{dB-1} \cdot nB + a2 \cdot X^{dB-2} \cdot nB^2 - \cdots - ai \cdot nB^{dB}) \bmod nB = X^{dB} \bmod nB - a1 \cdot X^{dB-1} \cdot nB \bmod nB + a2 \cdot X^{dB-2} \cdot nB^2 \bmod nB - \cdots - ai \cdot nB^{dB} \bmod nB$$

この式中の第２項以降は、全てｎＢで割り切れるので、削除できる。よって、

$$= X^{dB} \bmod nB \cdots (13)$$

となる。先に$CA^{eB}=X$としたので、元に戻すと、

$$= (CA^{eB})^{dB} \bmod nB \cdots (14)$$

が得られる。

【００１７】ここまでの過程をまとめると、

$$cA^{dB} \bmod nB \equiv (CA^{eB} \bmod nB)^{dB} \bmod nB = (CA^{eB})^{dB} \bmod nB$$

となる。

【００１８】さて、上記のｅＢ、ｄＢは、式（５）を満◆足するので、ある整数ｈを用いて

$$eB \cdot dB = h(pB-1)+1$$

と表せる。ここで、素数ｐと、ｐと互いに素な任意の整数Ｘに対して

$$X^{p-1} \bmod p = 1$$

が成立するというフェルマの小定理を用いると、

$$CA^{eB \cdot dB} \bmod pB = CA^{h(p-1)+1} \bmod pB = CA \cdot CA^{h(p-1)} \bmod pB = CA \bmod pB \cdots (15)$$

$$= MA$$

が導き出される。

となる。ＣＡがｐＢの倍数でも上式を満足するので、全てのＣＡについて$CA^{eB \cdot dB}-CA$は、ｐＢで割り切れる。同様に$CA^{eB \cdot dB}-CA$はｑＢでも割り切れる。ｐＢとｑＢは、異なる素数なので、$CA^{eB \cdot dB}-CA$はｎＢ＝ｐＢ・ｑＢでも割り切れる。よって、

$$cA^{dB} \bmod nB \equiv CA^{eB \cdot dB} \bmod nB \equiv CA \bmod nB \ (=CA)$$

が成立する。

【００１９】・送信者の公開鍵ｅＡを使用して、平文ＭＡを作成する。（署名認証）

$$CA^{eA} \bmod nA \equiv (MA^{dA})^{eA} \bmod nA \equiv (MA^{dA})^{eA} \bmod nA$$

上記した復号化処理と同様にして計算すると、

※10

※Ｘ＝ｋ・ｎＢ＋Ｙ

$$Y = X - k \cdot nB \cdots (10)$$

と変形できる。よって、式（１０）を式（９）の右辺に代入すると、

【００２０】以上のように、ｅ、ｄ、ｎの値を式（４）（５）のような条件下で決定し、式（１）のようなべき乗剰余演算の形式を基本として使用することにより、平文を暗号化しまた暗号化された平文を復号化することができる。

【００２１】例えば、ｎ＝１５、ｅ＝３、ｐ＝５、ｑ＝３、ｄ＝１１とし、

(n＝p・q＝5・3＝15、
e・dmod（p－1）・（q－1）＝3・11mod4・2
＝33mod8
＝1)

平文M＝13とすると、
暗号化：C＝M^emod n＝13^3mod 15＝2197 mod 15＝7
復号化：M＝C^dmod n＝7^{11}mod 15＝1977326743mod 15＝13
となり、平文M＝13が復号化されたことが確認された。

【0022】（3）べき乗剰余演算の演算方法
次に、暗号化・復号化で使用される、べき乗剰余演算の演算方法を説明する。

【0023】A＝M^emod Nのべき乗剰余演算は、整数eの2進数展開をe＝e^{k-1}・・・e^1e^0として、以下のフロー1に示す反復平方積法を使用して実行される。

【0024】

〔フロー1〕
```
begin
    A=1
    for  i=k-1 down to 0 do
    begin
        A=A²modN  ・・・（16）
        if ei=1 then A=A·MmodN ・・・（17）
    end
end
```

そして、この反復平方積法をフローチャートで表現すると、図3のようになる。

【0025】まず、初期値1をAレジスタに格納する。次に、Aレジスタに格納された値を乗算してA×Aを求め、このA×AをNで除算して余りを求め、この値をレジスタaに格納する。次に、レジスタaに格納された値をレジスタAに格納する。このとき、もし指数eが1ならば、レジスタAに格納された値と平文Mを乗算した後Nで除算して余りを求め、この値をレジスタaに格納する。そして、このレジスタaの内容を再度レジスタAに格納する。もし、指数eが0ならば、以上の計算は行わずに、レジスタAに格納された値には何も施さない。この演算をeの最上位ビットから最下位ビットまで繰り返し行い、最終的にレジスタAに格納された値が、求めたいべき乗剰余演算の解になる。

【0026】以上のように、演算の基本は、式（16）（17）に示すように乗算と除算（mod算）である。乗算は、初期値を1とするAの値に対してA×Aあるいは A×Mを行う部分であり、除算は、各々の乗算で得られた値に対してmod Nを行う部分である。この”乗算と除算”（A×Amod N、A×Mmod N）を一対の演算として、”e”のビット値に従って繰り返す。つまり、”e”の最上位ビットから最下位ビットまでの各ビットの内容によって”乗算と除算”を行うのである。

【0027】【発明が解決しようとする課題】べき乗剰余演算は、基本となる剰余演算（mod算）を繰り返し行うことで解を得ることができることを示したが、この繰り返し回数自体は、たかだか数百～数千回であるので、ソフト的な処理でも充分に対応できる。しかしながら、この剰余演算自体すなわち除算を実行するためには、大規模な演算回路と複雑な処理手順が必要とされるため改善が望まれていた。

【0028】【課題を解決するための手段】本発明の代表的なものは、正の整数M、e、Nに関するM^e mod Nなるべき乗剰余演算を行うべき乗剰余演算回路において、Nと素であり、かつNよりも大きい整数Rを用い、第1のモード信号に対応したタイミング制御信号に応答して、A・A・R’ mod N（R’はmod N法下でのRのインバースである。）なる第1の演算を実行し、前記整数Rを用い、第2のモード信号に対応したタイミング制御信号に応答して、A・B・R’ mod Nなる第2の演算を実行し、前記整数Rを用い、第3のモード信号に対応したタイミング制御信号に応答して、A・1・R’ mod Nなる第3の演算を実行する演算部と、前記第1ないし第3のモード信号を受信し、前記第1ないし第3のモード信号に対応した前記タイミング制御信号を前記演算部に出力するタイミング制御回路とを設けたものである。

【0029】【発明の実施の形態】べき乗除余演算は、基本となる剰余演算（mod算）を実行する手順が非常に複雑であるため、演算回路が大規模化してしまうことを前述した。そこで、Montgomery（モンゴメリー）は、剰余演算（mod算）を先のような一般的な方法で行わずに、”乗算”と簡単なビット列処理を行うことによって、剰余演算の解を得る仕組みを提案している。本発明は、基本的にこのモンゴメリーが提案している仕組みを利用して演算を行うものであるので、以下にモンゴメリーが提案している手法について簡単に説明するが、各演算において、演算時間を短縮するための工夫は、本願特有のものであることを述べておく。

【0030】剰余演算において、Rをモジュラス Nより少しだけ大きい2の指数と定義し、”乗算mod N”〔（ある値×ある値）÷（値N）の余りを求めるという意味。〕の法下でのRのインバース値をR’と定義し、（R・R’ mod N＝1が成立する。）さらに、R・R’－N・N’＝1、0＜N’＜Rなる関係を満足する

9

N’（N’は、”乗算ｍｏｄN”の法下でのNのインバース値である。）を定義する。この時に、例えば、M（X）＝XｍｏｄN形式の剰余演算を行う場合に、

M’（X）＝X・R’ｍｏｄN・・・（１８）

の形に置き換えて、次のフロー２に示すＲＥＤＣ（X）関数の計算方法を実行すれば、剰余演算（ｍｏｄ算）の解を先の一般的方法（乗算と除算とを単に実行するもの）によらずに得ることができるというものである。ただし、フロー２は剰余演算の解を求めるフローであり、べき乗剰余演算の解を求めるフローではない。なお、上記関数で得られたｔかｔ－Nが式（１８）の解である。

【００３１】

〔フロー２〕

ＲＥＤＣ（X）関数

```
function REDC(X)
 m=(XmodR)･N' modR        ･･･（１９）
 t=(X+m･N)/R              ･･･（２０）
   if  t<N
       return t           ･･･（２１）
   else  return t-N       ･･･（２２）
```

10

＊乗算と、Ｒを用いた除算で構成されている。Ｒ＝2^nという定義なので、Ｒを用いた除算は、被除算値の2^n以上の値が商、2^n未満の値が余りを示す。従って、式（１９）におけるｍｏｄ算においては、基本的に2^n未満の値を見るだけでよく、式（２０）における除算においては、基本的に2^n以上の値を見るだけでよい。つまり、実質的に除算（剰余演算）を行うことなく乗算と加算のみで剰余演算の解を求めることができる。

【００３２】次に参考のために、m、Ｒ、Ｎとの間の関係を式（１９）（２０）を使用して説明する。式（１９）において、

20

以上の関数を見ると、演算要素がN及びN’を使用した＊

$$m\cdot N=((X\bmod R)\cdot N'\bmod R)\cdot N$$
$$\equiv X\cdot N\cdot N'\bmod R$$
$$\equiv X\cdot(R\cdot R'-1)\bmod R$$
$$\equiv X\cdot R\cdot R'\bmod R-X\bmod R$$
$$\equiv -X\bmod R$$

$$X+m\cdot N\equiv X+(-X\bmod R)$$
$$\equiv 0\bmod R$$

となる。この式は、X＋ｍ・ＮをＲで割ったときの余りが０、言い替えると、X＋ｍ・Ｎは、Ｒで割り切れることを意味する。

【００３３】ここで、X＋ｍ・Ｎは、”X”と”Ｎの乗算”との和であるから、

（X＋ｍ・Ｎ）ｍｏｄN≡XｍｏｄN＋ｍ・NｍｏｄN≡XｍｏｄN

よって、式（２０）より、

ｔ・RｍｏｄN≡XｍｏｄN

この両辺をR’倍すると、

ｔ・Ｒ・R’ｍｏｄN≡ｔｍｏｄN≡X・R’ｍｏｄN

となる。

【００３４】ところで、上記式X・R’ｍｏｄN中のXが、ｍｏｄNあるいはｍｏｄＲを計算した後の乗算値であるならば、（Xが、ｍｏｄNあるいはｍｏｄＲを計算した後の乗算値であることは後に説明する。）

X＜N・N＜Ｒ・N＜Ｒ・Ｒ

mはｍｏｄＲを実行した結果であるから、m＜Ｒである。よって、ｍ・N＜Ｒ・Ｎとなる。

【００３５】X＜Ｒ・Ｎであるから、X＋ｍ・N＜Ｒ・N＋Ｒ・Ｎ＝２Ｒ・Ｎである。従って、

ｔ＝（X＋ｍ・Ｎ）／Ｒ＜２N

ｔ＜２N

が成立する。

【００３６】そして、もしｔがＮより大きい値の場合、両辺からＮを引くと、

ｔ－N＜N

となり、この式からｔ－Nは、ｍｏｄＮを施した値となることがわかる。

【００３７】さらに、式（２０）のｔ＝（X＋ｍ・Ｎ）／Ｒの計算と、ｔの計算終了について補足する。

【００３８】X＋ｍ・Ｎは、必ずＲの倍数であるから、例えば、Ｒ＝2^{576}とすると、X＋ｍ・Ｎの５７６ビット未満の値は全て０である。従って、式（２０）のｔの計算は次の２通りに分けられる。

【００３９】・XがＲの倍数でない場合

X＋ｍ・ＮとＲのビット列イメージを示すと以下のようになる。

【００４０】Ｒ＝１０００・・・０００

X＝　？？？？？？？・・・？？？？

ｍ・Ｎ＝　？？？？？？？・・・？？？？

X＋ｍ・Ｎ＝？？？？？？０００・・・０００

式（２０）の計算において、ｔは必ず整数値が得られるので、X＋ｍ・NはＲの倍数になる。従って、X＋ｍ・

Nは、上記ビット列イメージのようになる。そして、アンダーラインを付与した部分がｔの解となる。その理由は、X＋m・NはRの倍数であるから、X及びm・NのR値未満の加算結果は必ず0になるはずで、しかもそれらの？で示す値は元々オール0ではないので、加算の課程でR値以上の桁への桁上げが生じているはずである。よって、この場合、X及びm・N値のR値未満の値を無視して、

ｔ＝（X値のR以上の値）＋（m・N値のR値以上の値）＋1

の計算で解が得られる。

【0041】・XがRの倍数である場合

XとRのビット列イメージを示すと以下のようになる。

【0042】R＝1000・・・0000

X＝　？？？？？000・・・0000

このとき、式（19）中のXmodRは0となるので、m＝0となる。m＝0ならば式（20）の計算は、ｔ＝X／Rで良いことになる。ここで、ｔは必ず整数値が得られるので、

ｔ＝（X値のR以上の値）

の計算で解が得られる。

【0043】X値がRの倍数であるか否かの判別は、X値のR未満の値が0か否かで判別できる。このことは、ｔの計算においてX値の全ビット長分を計算する必要がなく、計算量の縮小と計算時間の短縮が図れることを意味する。

【0044】ｔの計算終了に関しては、REDC（X）関数の演算結果ｔは0＜ｔ＜2Nの範囲内の値となり、N＜ｔの時再度ｔ－Nの計算を行わなければならないことをフロー2で示した。しかし、べき乗剰余演算の途中で行われたREDC（X）関数の演算結果ｔについては、N＜ｔの関係にあってもｔ＜Rの範囲であれば、そのまま後の演算に進んでも構わない。なぜなら、後のmod算でここで取り残されたN値を除去するからである。

【0045】いま、べき乗剰余演算の途中の演算結果を＊

$$A\cdot A\cdot R'\bmod N=(N+2)\cdot(N+2)\cdot R'\bmod N$$
$$=(N^2+4N+4)\cdot R'\bmod N$$
$$\equiv N^2\cdot R'\bmod N+4N\cdot R'\bmod N+4\cdot R'\bmod N$$
$$\equiv 4\cdot R'\bmod N$$

となる。これは仮に前回のREDC（X）関数で得たｔの値がｔ＝N＋2ではなくmodN算の純粋な解ｔ＝2でA・A・R’ modNの計算を行なった場合と同じ結果となる。

【0050】（実際例）X＝44123，R＝2^8，N＝199の時のM’（X）＝X・R’ modNの計算。

【0051】R・R’ modN＝1、R・R’－N・N’＝1の関係からR’＝7、N’＝9。従って、M’（X）＝44123×7mod199＝13＝0D$_H$ となる筈である。ここでREDC（X）関数を使用して上記解を得る。

＊MmodN＝Sとすると、

M＝k1・N＋S＝（k1－1）・N＋N＋S

となる関係が成立しているから、今回のmod算でN＋Sを取り残したと考えると次のmod算、例えばべき乗剰余演算のA^2modN（図3参照）にて

M1＝A・A＝（N＋S）・（N＋S）

＝N^2＋2・N・S＋S^2

という処理を施して、再度mod算を行う。

【0046】

M1modN＝（N^2＋2・N・S＋S^2）modN

≡S^2modN

となり、これは前のmod算で取り残したN値を除去していることを意味する。また、M1をべき乗剰余演算のA・MmodNにて計算しても同様である。モンゴメリ手法下でも同様のことが言える。

【0047】簡単のために低ビット長の実際の数値例で説明する。

【0048】N＝13（1101），R＝（10000），前回のREDC（X）関数で得られた値ｔ＝15（1111）とすると、これらは、N≦ｔ＜R＜2Nの関係にある。この時のｔのbit4は”0”であるが、ｔの値自体未だmod算の純粋な解は得られていない。しかし、ｔはNのビット長を越えていないので、次のべき乗剰余演算のA・A・R’ modNあるいはA・M・R’ modNのべき乗剰余演算の上でAに代入する値としては妥当である。つまり、次に行なわれる乗算A・A、A・Mの結果の値が規定のビット長（上記の数値例では4ビット×2＝8ビット）を越えないので、ビット長が規定された”乗算R’ modN”の演算を継続して行なうことができる。

【0049】ところで、ｔ＝15＝13＋2＝N＋2と表せる。このことはmodNの計算の上では前回のREDC（X）関数で得たｔの値は”Nを取り残した値”であると言える。このｔの値のまま、Aに代入してA・A・R’ modNの計算を行なうと、

【0052】X＝44123＝AC5B$_H$，R＝100$_H$，N＝199＝C7$_H$，N’＝9$_H$より、

（REDC（X）関数）

m＝（XmodR）・N’ modR＝5B$_H$×9$_H$modR＝333$_H$modR＝33$_H$

ｔ＝（X＋m・N）／R＝（AC5B$_H$＋33$_H$×C7$_H$）／R＝D400$_H$／R＝D4$_H$

純粋な解は、ｔ－N＝D4$_H$－C7$_H$＝0D$_H$であるがこのままｔ・ｔ・R’ modNを実行してみる（次式1）

1、t＝$D4_H$→t・t＝$D4_H$×$D4_H$＝$AF90_H$の場合

m＝（XmodR）・N’ modR＝90_H×9_Hmod R＝510_HmodR＝10_H

t＝（X＋m・N）／R＝（$AF90_H$＋10_H×C 7_H）／R＝$\underline{BC}00_H$／R＝$\underline{BC}_H$

2、t＝$0D_H$→t・t＝$0D_H$×$0D_H$＝$00A9_H$の場合

m＝（XmodR）・N’ modR＝$A9_H$×9_Hmod R＝$5F1_H$modR＝$F1_H$

t＝（X＋m・N）／R＝（$00A9_H$＋$F1_H$×C 7_H）／R＝$\underline{BC}00_H$／R＝$\underline{BC}_H$

上式1、2の計算結果は、アンダーラインように同値となる。

【0053】このことは、演算時間短縮を図る上では重要である。つまり、Nの最上位ビットをbit(n-1)とするとき、得られたtのbit(n)が1の時t－Nを行い、0の時は後の演算に進むようにすれば良い。

【0054】一般に576ビット長オペランドのべき乗剰余演算の場合は、R＝2^{576}にする。そのとき、式（20）の計算結果が576ビット長を越えた場合（576ビットの桁あふれが生じた場合）が、t≧Rを示す。また、その場合のt－Nの計算は、

t－N＝（t値の下位576ビットの値）＋（Nのインバース値）＋1

で求める。暗号アルゴリズムの世界でのNの値は奇数であるから、Nのインバース値＋1は、Nのインバースを求めてその最下位ビットを1にする操作で得られる。

【0055】さて、フロー2で示した、モンゴメリー演＊

＊算を実行するためのREDC関数は、モンゴメリー演算の解を得ているにすぎない。すなわち、式（18）に示すように、計算を容易にするためR’という特有の数値を使用している。そもそも求めたいのは、R’を含まない乗算modN形式を使用した式の解であるので、フロー2で示すモンゴメリ演算の解を、何がしかの操作でR’を含まない数値に戻す必要がある。

【0056】そこで、本願では、次のような性質を利用して、モンゴメリ演算特有の上記R’という値を相殺した。

【0057】今、M’（X）＝X・R’ modNにおいて、Xが（X・R）であるとき、

M’（X・R）＝（X・R）R’ modN＝XmodN

Xが（X・RmodN）であるとき、

M’（X・RmodN）＝（X・RmodN）R’ modN＝XmodN

となる。

【0058】つまり、？・Rあるいは？・RmodNの形のものにモンゴメリ演算を適用すると、モンゴメリ演算特有のR’が除去されて、？modNの形になる。

【0059】そして、この性質をフロー1で示した、反復平方積法を使用した、べき乗剰余演算に応用すれば、フロー3で示すように、べき乗剰余演算の解が得られる。正確には、フロー3で示す式（24）（25）（26）のmodN算の各々において、フロー2を実行することにより、各式（24）（25）（26）における解が容易に得られ、結果としてフロー3を使用した、べき乗剰余演算の解が容易に得られる。

【0060】

〔フロー3〕

```
begin
    A=1･RmodN=RmodN  ・・・（23）
    for  i=k-1 down to 0 do
    begin
        A=A^2･R'modN  ・・・（24）
        if ei=1 then A=A･(M･RmodN)･R'modN ・・・（25）
    end
    A=A･R'modN  ・・・（26）
end
```

フロー1とフロー3との大きな違いは、上記式（23）において、レジスタAに格納する初期値を1とせず、後のモンゴメリ演算を考慮してRmodN（1・RmodN）を格納している点、上記式（25）において、前記式（17）のMの代わりにM・RmodNの値を用いている点及び上記式（26）を新たに行う点である。レジスタAにあらかじめRmodNを格納しておくことにより、式（24）（25）（26）におけるR’が除去されるのである。そして、フロー3の反復平方積法をフローチャートで表現すると、図1のようになる。

【0061】演算の実行に先だって、この図1において、あらかじめ必要とされるRmodN、M・RmodN、N’を求めておかなければならない。

【0062】〔RmodNについて〕RSA暗号等に用いられるべき乗剰余演算においては、”N”の最上位ビットbn-1と最下位ビットb0は1である。従って、そのときのRは、2^nに選ばれる。よって、RmodN＝R－Nとなる。このR－Nは、Nのインバースを求めて、その最下位ビットを1にすることで容易に求められる。

【0063】〔N’について〕R・R’ modN＝1、R・R’－N・N’＝1、0＜R’、N’＜Rの関係より、

15

$R' < N'$、$N' - R < R - N$

なる関係を導くことができる。

【００６４】暗号システムの運用者は、これらの関係を満足するように、N値に対応するN'値を決定する。す＊

$$M \cdot (R^2 \bmod N) \cdot R' \bmod N \equiv M \cdot R^2 \cdot R' \bmod N$$
$$\equiv M \cdot R \bmod N$$

により、$M \cdot R \bmod N$を求めることができる。

【００６６】例えば、今、$R = 2^{512}$とすると、

$2^{513} \bmod N = 2 \cdot 2^{512} \bmod N \equiv 2 \cdot (R \bmod N) \bmod N = A$

とすると、このAを初期値として以下のフロー４を実行すると$R^2 \bmod N$の解を求めることができる。

【００６７】

〔フロー4〕

```
begin
  for i=1 to 9
    begin M(A,A)=A･A･R' modN
    end
end
```

このフロー４中の計算課程は、以下の通りである。

$i=1$ $A = 2^{513} \bmod N$ $M(A,A) = 2^{513} \cdot 2^{513} \cdot 2^{-512} \bmod N \equiv 2^{514} \bmod N$

$i=2$ $A = 2^{514} \bmod N$ $M(A,A) = 2^{514} \cdot 2^{514} \cdot 2^{-512} \bmod N \equiv 2^{516} \bmod N$

$i=3$ $A = 2^{516} \bmod N$ $M(A,A) = 2^{516} \cdot 2^{516} \cdot 2^{-512} \bmod N \equiv 2^{520} \bmod N$

$i=4$ $A = 2^{520} \bmod N$ $M(A,A) = 2^{520} \cdot 2^{520} \cdot 2^{-512} \bmod N \equiv 2^{528} \bmod N$

$i=5$ $A = 2^{528} \bmod N$ $M(A,A) = 2^{528} \cdot 2^{528} \cdot 2^{-512} \bmod N \equiv 2^{544} \bmod N$

$i=6$ $A = 2^{544} \bmod N$ $M(A,A) = 2^{544} \cdot 2^{544} \cdot 2^{-512} \bmod N \equiv 2^{576} \bmod N$

$i=7$ $A = 2^{576} \bmod N$ $M(A,A) = 2^{576} \cdot 2^{576} \cdot 2^{-512} \bmod N \equiv 2^{640} \bmod N$

$i=8$ $A = 2^{640} \bmod N$ $M(A,A) = 2^{640} \cdot 2^{640} \cdot 2^{-512} \bmod N \equiv 2^{768} \bmod N$

$i=9$ $A = 2^{768} \bmod N$ $M(A,A) = 2^{768} \cdot 2^{768} \cdot 2^{-512} \bmod N \equiv 2^{1024} \bmod N$

そして、最後の結果（i=9）が求めたい$R^2 \bmod N$の解である。

【００６９】ここで求めた$R^2 \bmod N$とMとを乗算した後、この乗算結果をフロー２におけるXとして、フロー２を実行することにより、求めたい$M \cdot R \bmod N$が得られる。

【００７０】前述した通り、フロー２における演算は、N及びN'を使用した乗算と、Rを用いた除算で構成されている。従って、この解は、実質的に除算（剰余演算）を行うことなく乗算と加算のみで求めることができる。

【００７１】以上、図１のフローチャートにおいてあらかじめ必要とされる$R \bmod N$、$M \cdot R \bmod N$、N'の値が用意できた。

【００７２】次に、図１のフローチャートを実行する。

【００７３】まず、あらかじめ求めた$R \bmod N$を初期値としてレジスタA（レジスタは、例えばメモリであっても良い。以下同様である。）に格納する。そして、あらかじめ求めた$M \cdot R \bmod N$をレジスタBに格納する。$M \cdot R \bmod N$をレジスタBに格納する理由は、後の処理で使用する式（２５）中の$M \cdot R \bmod N$を確保

16

＊なわち、N'値は運用者から与えられるものである。

【００６５】〔$M \cdot R \bmod N$について〕あらかじめ$R^2 \bmod N$の値を求めておくと、モンゴメリ演算を使用した以下の計算

$$M \cdot (R^2 \bmod N) \cdot R' \bmod N \equiv M \cdot R^2 \cdot R' \bmod N$$
$$\equiv M \cdot R \bmod N$$

※【００６８】

10

20

30

しておくためである。

【００７４】次に、式（２４）を実行する。すなわちモンゴメリ演算$A^2 \cdot R' \bmod N$を実行する。式（２４）は、前述したようにフロー２を実行することにより得られる。今、レジスタAに格納された値を２乗した値が式（１９）におけるXとして計算され、mが求められる。前述のようにこの演算は、乗算とRを用いた剰余演算によって実行される。$R = 2^n$という定義なので、Rを用いた剰余演算は、被除算値の2^n以下の値を見るだけでよい。次に、求められたmを用いて、式（２０）を

40 実行する。今、レジスタAに格納された値を２乗した値が式（２０）におけるXとして計算され、tが求められる。前述のようにこの演算は、乗加算とRを用いた除算によって実行される。$R = 2^n$という定義なので、Rを用いた除算は、被除算値の2^n以上の値を見るだけでよい。そしてこの演算結果をレジスタAに格納する。

【００７５】次に、レジスタaに格納された値をレジスタAに格納する。

【００７６】ここで、指数eのビットの判別を行う。もし、ビットが１ならば式（２５）を実行する。すなわち

50 モンゴメリ演算$A \cdot B \cdot R' \bmod N$を実行する。式

（２５）は、前述したようにフロー２を実行することにより得られる。今、レジスタＡに格納された値とレジスタＢに格納された値との積が式（１９）におけるＸとして計算され、ｍが求められる。次に、求められたｍを用いて、式（２０）を実行する。今、レジスタＡに格納された値とレジスタＢに格納された値との積が式（２０）におけるＸとして計算され、ｔが求められる。そしてこの演算結果をレジスタａに格納する。そして、レジスタａに格納された演算結果をレジスタＡに格納する。

【００７７】もし指数ｅのビットが０ならば式（２５）の演算を行わずに次のステップに進む。

【００７８】次に、指数ｅの全てのビットに対して以上の演算（式（２４）（２５）の実行）を行ったかどうか判定し、行っていないのであれば式（２４）を実行するステップに戻り、行ったのであれば次に式（２６）を実行する。式（２６）は、前述したようにフロー２を実行することにより得られる。今、レジスタＡに格納された値が式（１９）におけるＸとして計算され、ｍが求められる。次に、求められたｍを用いて、式（２０）を実行する。今、レジスタＡに格納された値が式（２０）におけるＸとして計算され、ｔが求められる。そして、以上の一連の動作が終了する。

【００７９】以上のべき乗剰余演算の解法の概略をまとめると以下のようになる。

【００８０】

１、べき乗剰余演算（M^eｍｏｄＮ）

↓

２、反復平方積法を用いる。（A･AmodN、A･MmodNの繰り返し。）

↓

３、A･AmodN、A･MmodNの計算は、除算が複雑なためモンゴメリ手法に置き換える。（モンゴメリ手法：M'（Ｘ）＝Ｘ・Ｒ' ｍｏｄＮ）

↓

$A^2 \cdot R' \bmod N \cdots (24)$、$A \cdot B \cdot R' \bmod N \cdots (25)$

（Ｂは、式（２５）における”Ｍ・ＲｍｏｄＮ”が位置する部分に相当する。）の２種の演算を、指数”ｅ”のビット値の内容に従って、規定の手順で繰り返し最後に、

Ａ・１・Ｒ' ｍｏｄＮ・・・（２６）

の演算を行えば良いことは、図１及びフロー３で述べたとおりである。

【００８６】従って、上記３種類の剰余演算を実行する演算器（以下、コプロセサと称す。）を用意すれば、そのコプロセッサを用いて（演算の繰り返し手順等は、ソフトウエアあるいはハードウエアのいずれの制御方法で実現しても良い。）、べき乗剰余演算の解を得ることができる。

【００８７】本発明は、上記の３種類の剰余演算、すなわち３つの演算モードを有するコプロセッサに関するも

＊４、Ｍ'（Ｘ）＝Ｘ・Ｒ' ｍｏｄＮは、ＲＥＤＣ（Ｘ）関数をしようして実現できる。（ＲＥＤＣ関数：ｍｏｄＮの演算をｍｏｄＲの形に変換するもの。これにより除算の複雑性を回避できる。）

次に、本発明を実際に実現するためのハードウエアについて、図面を使用して以下に説明する。

【００８１】まず、実際のハードウエアの概略を示すブロック図を図４に示す。

【００８２】図４において、４０１、４０３、４０５、４０７、４０９、４１１、４１３、４１５は、メモリもしくはレジスタである。そして、４０１、４０３、４０５、４０７、４０９、４１１、４１３には、各々ブロック図中に記載された値が格納されている。また、レジスタ４０５は、図１中で説明したレジスタＡに相当するものであり、レジスタ４１５は、レジスタａに相当するものである。セレクタ４１７は、指数ｅが格納された１ビット左シフトレジスタ４０９からの指示に基づき、レジスタ４０７もしくはレジスタ４０５の出力のいずれかを演算部４１９中の乗算部に転送する。詳しくは、セレクタ４１７は、図１中の”ｅシフトキャリ”を実行する部分、すなわち指数部ｅが１の場合はＭ・ＲｍｏｄＮを演算部に転送し、指数部ｅが０の場合は転送しない。また、図１中のフローの初期段階において、A^2を求めるときには、レジスタ４０５の出力を演算部に転送する。

【００８３】演算部４１９は、乗算部と除算部とで構成されている。この演算部４１９は、乗算とＲＥＤＣ（Ｘ）関数を実行する。

【００８４】次に、本発明を実際に実現するためのハードウエアの詳細について、図面を使用して以下に説明する。

【００８５】（第１の実施の形態）

（構成）べき乗剰余演算M^eｍｏｄＮの計算方法は、先に示したモンゴメリ演算手法における、

＊

のである。

【００８８】図５は、本発明の第１の実施の形態の剰余演算コプロセッサを示す図である。

【００８９】なお、図面中において、各ブロックを結ぶ太い矢印は、データが転送されるバスを示す。

【００９０】本発明の剰余演算コプロセッサは、コプロセッサ全体の演算タイミング及び演算器内の各種回路に対して、上記３種類の演算の種類に対応したコントロール信号を供給するタイミング／コントロール回路Ｔ／Ｃと、モンゴメリ法化における各演算値を格納する、複数の演算値メモリＳｍｅｍ、Ｎ' ｍｅｍ、Ｎｍｅｍ、Ｍｍｅｍ、Ａ' ｍｅｍ、Ｗｌｍｅｍ、Ｗｈｍｅｍを有する。また、本発明の剰余演算コプロセッサは、乗加算を行う高速乗加算器Ｍｕｌ／Ａｄｄ、加算を行う高速加算器Ａｄｄ、乗算値を格納する乗算値格納レジスタＸｉ－ｒｅ

g、被乗算値を格納する被乗算値格納レジスタＹｉ－ｒｅｇ、被加算値を格納する被加算値格納レジスタＡｉ－ｒｅｇ、高速加算器Ａｄｄが出力する値の上位の桁を格納するレジスタＲＨを有する。

【００９１】高速加算器Ａｄｄ、乗算値格納レジスタＸｉ－ｒｅｇ、被乗算値格納レジスタＹｉ－ｒｅｇ、被加算値格納レジスタＡｉ－ｒｅｇは、各演算値メモリから読み出された、高速乗加算器Ｍｕｌ／Ａｄｄの入力ビット長分に対応した値を一時格納する機能を有する。

【００９２】高速乗加算器Ｍｕｌ／Ａｄｄは、乗算動作の際の入力値として、レジスタＸｉ－ｒｅｇの出力値と、レジスタＹｉ－ｒｅｇの出力値を入力し、加算動作の際の入力値として、レジスタＡｉ－ｒｅｇの出力値を入力する、特定ビット長の乗加算器である。そして、その出力は、次段の高速加算器Ａｄｄの加算入力値となる。

【００９３】高速加算器Ａｄｄは、高速乗加算器Ｍｕｌ／Ａｄｄの出力値と、レジスタＲＨの出力値との加算動作を行う、特定ビット長の加算器である。そして、この高速加算器Ａｄｄの出力のうち、上位桁はレジスタＲＨもしくは各演算値メモリに向けて出力され、下位桁は各演算値メモリに向けて出力される。

【００９４】（動作）次に、図５に示す回路の動作を説明する。

【００９５】〔$A^2 \cdot R'$ ｍｏｄＮ・・・（２４）の実現の方法〕Ａの値を演算値メモリＡ’ ｍｅｍ、Ｗｌｍｅｍの双方に格納し、モンゴメリ法化におけるＮ’の値を演算値メモリＮ’ ｍｅｍに格納し、Ｎの値を演算値メモリＮｍｅｍに格納する。Ｎ及びＮ’の値は、暗号化／復号化のための鍵であるので、データの送信者／受信者に対して暗号システム運用者により決定されている。Ｒの値は、Ｎの値から自ずと決定される。Ａの値は、前述したように、ＲｍｏｄＮを実行することで決定される。

【００９６】そして、タイミング／コントロール回路Ｔ／Ｃに対して、モード１信号を供給することにより、以下のように上記$A^2 \cdot R'$ ｍｏｄＮ・・・（２４）が実行される。

【００９７】まず、Ａ×Ａの計算が行われる。

【００９８】演算値メモリＡ’ ｍｅｍから高速乗加算器Ｍｕｌ／Ａｄｄの入力ビット長分に対応した値が、レジスタＸｉ－ｒｅｇに取り込まれる。同様に、演算値メモリＷｌｍｅｍから高速乗加算器Ｍｕｌ／Ａｄｄの入力ビット長分に対応した値が、レジスタＹｉ－ｒｅｇに取り込まれる。

【００９９】図６に示すように、乗算動作を行う場合に、演算値のビット長（例えば１６ビット）が高速乗加算器Ｍｕｌ／Ａｄｄの乗算入力処理ビット長（例えば４ビット）を越えているとき、演算値のビット長に対応した演算の回数分（４×４＝１６回）だけ単位乗加算を繰り返すことになる。

【０１００】上述の動作をハードウエアイメージでより詳細に表現すると図７のようになる。

【０１０１】この演算は、演算値メモリＡ’ ｍｅｍの各アドレスＡ’３、Ａ’２、Ａ’１、Ａ’０に格納された値と演算値メモリＷｌｍｅｍの各アドレスＷｌ３、Ｗｌ２、Ｗｌ１、Ｗｌ０に格納された値とを乗算したときの値が、演算値メモリＷｈｅｍ及び演算値メモリＷｌｍｅｍの各アドレスＷｈ３、Ｗｈ２、Ｗｈ１、Ｗｈ０、Ｗｌ３、Ｗｌ２、Ｗｌ１、Ｗｌ０に格納されることを意味する。

【０１０２】（単位乗加算１）まず、レジスタＹｉ－ｒｅｇが、演算値メモリＷｌｍｅｍのアドレスＷｌ０に格納された値（被乗算値）を取り込むとともに、レジスタＸｉ－ｒｅｇが演算値メモリＡ’ ｍｅｍのアドレスＡ’０に格納された値（乗算値）を取り込む。次に、高速乗加算器Ｍｕｌ／Ａｄｄはこの乗算値と被乗算値を乗算し、高速加算器Ａｄｄに向けて出力する。この単位乗加算１では、桁合わせ動作を行う必要がないため、レジスタＡｉ－ｒｅｇから高速乗加算器Ｍｕｌ／Ａｄｄへ与える加算値の値を０とする。（レジスタＡｉ－ｒｅｇの値を０とするか、レジスタＡｉ－ｒｅｇに格納された値が高速乗加算器Ｍｕｌ／Ａｄｄへ与えられないようにする。）さらに、高速加算器ＡｄｄのＴｈｒｏｕｇｈ端子に”ｈｉｇｈレベル”を供給して、レジスタＲＨの内容を加算しないようにする。そして、演算値メモリＡ’ ｍｅｍの値と演算値メモリＷｌｍｅｍの値との乗算において、単位乗加算１の演算結果の下位桁は、本乗算の最終演算結果の最下位桁であるので、この単位乗加算１の演算結果の下位桁は本乗算の最終演算結果として演算値メモリＷｌｍｅｍの最下位アドレスＷｌ０に格納される。（図７において、アンダーラインが付与されたアドレスには、最終演算結果が格納されていることを意味する。）この単位乗加算１の演算結果の上位桁は、次の単位乗加算の桁合わせのためにレジスタＲＨに格納される。

【０１０３】（単位乗加算２）レジスタＹｉ－ｒｅｇは、すでに格納された値（被乗算値）を維持し、レジスタＸｉ－ｒｅｇは、演算値メモリＡ’ ｍｅｍのアドレスＡ’１に格納された値（乗算値）を取り込む。次に、高速乗加算器Ｍｕｌ／Ａｄｄはこの乗算値と被乗算値を乗算し、高速加算器Ａｄｄに向けて出力する。この単位乗加算２では、レジスタＡｉ－ｒｅｇから高速乗加算器Ｍｕｌ／Ａｄｄへ与える加算値の値を０とする。（レジスタＡｉ－ｒｅｇの値を０とするか、レジスタＡｉ－ｒｅｇに格納された値が高速乗加算器Ｍｕｌ／Ａｄｄへ与えられないようにする。）が、高速加算器ＡｄｄのＴｈｒｏｕｇｈ端子に”ｌｏｗレベル”を供給して、レジスタＲＨの内容（単位乗加算１の演算結果の上位桁）を高速乗加算器Ｍｕｌ／Ａｄｄの出力に加算する。この単位乗加算２の演算結果の下位桁は、演算値メモリＷｈｍｅｍ

の最下位アドレスＷｈ０に格納される。この単位乗加算２の演算結果の上位桁は、次の単位乗加算の桁合わせのためにレジスタＲＨに格納される。

【０１０４】（単位乗加算３）レジスタＹｉ－ｒｅｇは、すでに格納された値（被乗算値）を維持し、レジスタＸｉ－ｒｅｇは、演算値メモリＡ’ｍｅｍのアドレスＡ’２に格納された値（乗算値）を取り込む。次に、高速乗加算器Ｍｕｌ／Ａｄｄはこの乗算値と被乗算値を乗算し、高速加算器Ａｄｄに向けて出力する。この単位乗加算３では、レジスタＡｉ－ｒｅｇから高速乗加算器Ｍｕｌ／Ａｄｄへ与える加算値の値を０とする。（レジスタＡｉ－ｒｅｇの値を０とするか、レジスタＡｉ－ｒｅｇに格納された値が高速乗加算器Ｍｕｌ／Ａｄｄへ与えられないようにする。）が、高速加算器ＡｄｄのＴｈｒｏｕｇｈ端子に”ｌｏｗレベル”を供給して、レジスタＲＨの内容（単位乗加算１の演算結果の上位桁）を高速乗加算器Ｍｕｌ／Ａｄｄの出力に加算する。この単位乗加算３の演算結果の下位桁は、演算値メモリＷｈｍｅｍのアドレスＷｈ１に格納される。この単位乗加算３の演算結果の上位桁は、次の単位乗加算の桁合わせのためにレジスタＲＨに格納される。

【０１０５】（単位乗加算４）レジスタＹｉ－ｒｅｇは、すでに格納された値（被乗算値）を維持し、レジスタＸｉ－ｒｅｇは、演算値メモリＡ’ｍｅｍのアドレスＡ’３に格納された値（乗算値）を取り込む。次に、高速乗加算器Ｍｕｌ／Ａｄｄはこの乗算値と被乗算値を乗算し、高速加算器Ａｄｄに向けて出力する。この単位乗加算４では、レジスタＡｉ－ｒｅｇから高速乗加算器Ｍｕｌ／Ａｄｄへ与える加算値の値を０とする。（レジスタＡｉ－ｒｅｇの値を０とするか、レジスタＡｉ－ｒｅｇに格納された値が高速乗加算器Ｍｕｌ／Ａｄｄへ与えられないようにする。）が、高速加算器ＡｄｄのＴｈｒｏｕｇｈ端子に”ｌｏｗレベル”を供給して、レジスタＲＨの内容（単位乗加算１の演算結果の上位桁）を高速乗加算器Ｍｕｌ／Ａｄｄの出力に加算する。この単位乗加算４の演算結果の下位桁は、演算値メモリＷｈｍｅｍのアドレスＷｈ２に格納される。この単位乗加算４の演算結果の上位桁は、次の単位乗加算の桁合わせのために演算値メモリＷｈｍｅｍのアドレスＷｈ３に格納される。

【０１０６】（単位乗加算５）レジスタＹｉ－ｒｅｇは、演算値メモリＷｌｍｅｍのアドレスＷｌｌに格納された値（被乗算値）を取り込み、レジスタＸｉ－ｒｅｇは、演算値メモリＡ’ｍｅｍのアドレスＡ’０に格納された値（乗算値）を取り込み、レジスタＡｉ－ｒｅｇは、演算値メモリＷｈｍｅｍのアドレスＷｈ０に格納された値（加算値）を取り込む。次に、高速乗加算器Ｍｕｌ／Ａｄｄはこの乗算値と被乗算値を乗算し、その乗算結果に加算値を加算し、高速加算器Ａｄｄに向けて出力する。この単位乗加算５では、高速加算器ＡｄｄのＴｈｒｏｕｇｈ端子に”ｈｉｇｈレベル”を供給して、レジスタＲＨの内容（単位乗加算４の演算結果の上位桁）を加算しないようにする。この単位乗加算５の演算結果の下位桁は、演算値メモリＷｌｍｅｍのアドレスＷｌｌに格納される。この単位乗加算５の演算結果の上位桁は、次の単位乗加算の桁合わせのためにレジスタＲＨに格納される。

【０１０７】（単位乗加算６）レジスタＹｉ－ｒｅｇは、すでに格納された値（被乗算値）を維持し、レジスタＸｉ－ｒｅｇは、演算値メモリＡ’ｍｅｍのアドレスＡ’１に格納された値（乗算値）を取り込み、レジスタＡｉ－ｒｅｇは、演算値メモリＷｈｍｅｍのアドレスＷｈ１に格納された値（加算値）を取り込む。次に、高速乗加算器Ｍｕｌ／Ａｄｄはこの乗算値と被乗算値を乗算し、その乗算結果に加算値を加算し、高速加算器Ａｄｄに向けて出力する。この単位乗加算６では、高速加算器ＡｄｄのＴｈｒｏｕｇｈ端子に”ｌｏｗレベル”を供給して、レジスタＲＨの内容（単位乗加算５の演算結果の上位桁）を高速乗加算器Ｍｕｌ／Ａｄｄの出力に加算する。この単位乗加算６の演算結果の下位桁は、演算値メモリＷｈｍｅｍのアドレスＷｈ０に格納される。この単位乗加算６の演算結果の上位桁は、次の単位乗加算の桁合わせのためにレジスタＲＨに格納される。

【０１０８】（単位乗加算７）レジスタＹｉ－ｒｅｇは、すでに格納された値（被乗算値）を維持し、レジスタＸｉ－ｒｅｇは、演算値メモリＡ’ｍｅｍのアドレスＡ’２に格納された値（乗算値）を取り込み、レジスタＡｉ－ｒｅｇは、演算値メモリＷｈｍｅｍのアドレスＷｈ２に格納された値（加算値）を取り込む。次に、高速乗加算器Ｍｕｌ／Ａｄｄはこの乗算値と被乗算値を乗算し、その乗算結果に加算値を加算し、高速加算器Ａｄｄに向けて出力する。この単位乗加算７では、高速加算器ＡｄｄのＴｈｒｏｕｇｈ端子に”ｌｏｗレベル”を供給して、レジスタＲＨの内容（単位乗加算６の演算結果の上位桁）を高速乗加算器Ｍｕｌ／Ａｄｄの出力に加算する。この単位乗加算７の演算結果の下位桁は、演算値メモリＷｈｍｅｍのアドレスＷｈ１に格納される。この単位乗加算７の演算結果の上位桁は、次の単位乗加算の桁合わせのためにレジスタＲＨに格納される。

【０１０９】（単位乗加算８）レジスタＹｉ－ｒｅｇは、すでに格納された値（被乗算値）を維持し、レジスタＸｉ－ｒｅｇは、演算値メモリＡ’ｍｅｍのアドレスＡ’３に格納された値（乗算値）を取り込み、レジスタＡｉ－ｒｅｇは、演算値メモリＷｈｍｅｍのアドレスＷｈ３に格納された値（加算値）を取り込む。次に、高速乗加算器Ｍｕｌ／Ａｄｄはこの乗算値と被乗算値を乗算し、その乗算結果に加算値を加算し、高速加算器Ａｄｄに向けて出力する。この単位乗加算８では、高速加算器ＡｄｄのＴｈｒｏｕｇｈ端子に”ｌｏｗレベル”を供給して、レジスタＲＨの内容（単位乗加算７の演算結果の

上位桁）を高速乗加算器Ｍｕｌ／Ａｄｄの出力に加算する。この単位乗加算８の演算結果の下位桁は、演算値メモリＷｈｍｅｍのアドレスＷｈ２に格納される。この単位乗加算８の演算結果の上位桁は、演算値メモリＷｈｍｅｍのアドレスＷｈ３に格納される。

【０１１０】単位乗加算９から１６までは、前述の単位乗加算に習って、図７のごとく実行される。そして、Ａ×Ａすなわち、演算値メモリＡ’ｍｅｍにに格納された値と演算値メモリＷｌｍｅｍに格納された値とを乗算したときの値が、Ｗｈｍｅｍ－Ｗｌｍｅｍ（”－”は、引き算を示すものではない。）の形で格納される。

【０１１１】例えば、８ビット×８ビットの計算、６４Ｈ×Ｆ５Ｈ＝６５７２Ｈを例にとると、計算結果６５７２Ｈについて、上位桁６５Ｈを演算値メモリＷｈｍｅｍに、下位桁７２Ｈを演算値メモリＷｌｍｅｍにそれぞれ格納する。

【０１１２】次に、ＲＥＤＣ関数を使用して、モンゴメリ手法を用いた剰余演算を実行する。モンゴメリ手法において、オペランド長ｎビットのべき乗剰余演算を実行するとき、Ｒ＝2^nとすることは前述した通りである。そして、この関係でのｎビット長のＲ’、Ｎ’の値は存在する。そして、Ａ×Ａの値は、Ｗｈｍｅｍ－Ｗｌｍｅ＊

$$t = (X + m \cdot N) / R$$

$$= [(Whmem - Wlmem) + (Mmem) \cdot (Nmem)] / R$$

上式の解を得るためには、まず（Ｍｍｅｍ）×（Ｎｍｅｍ）の計算を先のＡ×Ａに習って実行し、その結果を

$$Mmem \times 1 + Wlmem \cdots (27)$$

$$A'mem \times 1 + Whmem \cdots (28)$$

の計算を実行する。このときの式（２８）が求めるｔである。

【０１１８】また、Ｒ以下の値は必ず０であるから、式（２７）の計算結果０となるので、メモリに格納する必要はない。なお、式（２７）の計算では、桁あふれが発生する場合が考えられるが、そのときには、式（２８）の計算の際に、図５に示した高速乗加算器Ｍｕｌ／Ａｄｄの＋１端子に、１を供給する。これらの計算は、本コプロセッサを用いれば容易に実行することができるのは★

$$[(\text{反転}Nmem + 1) \times 1] + Wlmem \cdots (29)$$

の計算を行って求める。反転Ｎｍｅｍは、Ｎｍｅｍに格納された値のインバースを意味する。

【０１２１】ここで、演算値メモリＮｍｅｍに格納されているＮの値は、暗号アルゴリズムの世界では奇数であるから、式（２９）の（反転Ｎｍｅｍ＋１）は”Ｎｍｅｍの１の補数をとり、最下位ビットを１とする”ことで求められる。この動作は、図５に示すレジスタＸｉ－ｒｅｇのＩｎｖ端子に１を供給する（レジスタＸｉ－ｒｅｇに格納された値の１の補数をとる。）ことで実現される。

【０１２２】〔Ａ・Ｂ・Ｒ’ｍｏｄＮ・・・（２５）の実現の方法〕Ａの値を演算値メモリＷｌｍｅｍに、Ｂの＊ｍの形で格納されているので、ｍの計算は以下のようになる。

【０１１３】

$$m = (X \bmod R) \cdot N' \bmod R$$

$$= (Whmem - Wlmem) \bmod R \cdot N' \bmod R$$

ここで、（Ｘｍｏｄ Ｒ）は、ＸをＲで割った余りをしめすので、Ｒ以下の値を見ればよいので、

$$= (Wlmem) \cdot (N'mem) \bmod R$$

となる。

【０１１４】そして、上式は（Ｗｌｍｅｍ）・（Ｎ’ｍｅｍ）をＲで割った余りを示すので、（Ｗｌｍｅｍ）・（Ｎ’ｍｅｍ）を実行した結果の下位ｎビットの値がｍとなる。

【０１１５】すなわち、（Ｗｌｍｅｍ）・（Ｎ’ｍｅｍ）の計算を、上述のＡ×Ａの計算に習って実行し、その結果をＡ’ｍｅｍ－Ｍｍｅｍの形で、演算値メモリＡ’ｍｅｍと演算値メモリＭｍｅｍの双方に格納したときの、演算値メモリＭｍｅｍに格納された値がｍである。

【０１１６】また、ｔの計算は以下のようになる。

【０１１７】

※Ａ’ｍｅｍ－Ｍｍｅｍの形で、演算値メモリＡ’ｍｅｍと演算値メモリＭｍｅｍの双方に格納する。続いて、

（下位桁の足し算）

（上位桁の足し算）

★明かである。

【０１１９】式（２８）で得られた値は、演算値メモリＡ’ｍｅｍとＭｍｅｍの双方に格納する。また、桁あふれが発生した場合は、キャリーフラグＣＦに１を書き込む。

【０１２０】ところで、キャリーフラグが１の場合は、ｔ＞Ｎの関係にあることは明かであるから、ｔ－Ｎの計算を行う必要がある。これは、

値を演算値メモリＳｍｅｍに格納し、モンゴメリ法化におけるＮ’の値を演算値メモリＮ’ｍｅｍに格納し、Ｎの値を演算値メモリＮｍｅｍに格納する。

【０１２３】そして、タイミング／コントロール回路Ｔ／Ｃに対して、モード２信号を供給することにより、以下のように上記Ａ・Ｂ・Ｒ’ｍｏｄＮ・・・（２５）が実行される。

【０１２４】まず、Ａ×Ｂの計算を行う。

【０１２５】演算値メモリＳｍｅｍから高速乗加算器Ｍｕｌ／Ａｄｄの入力ビット長分に対応した値が、レジスタＸｉ－ｒｅｇに取り込まれる。同様に、演算値メモリＷｌｍｅｍから高速乗加算器Ｍｕｌ／Ａｄｄの入力ビッ

ト長分に対応した値が、レジスタＹｉ－ｒｅｇに取り込まれる。

【０１２６】このＡ×Ｂの計算は、レジスタＸｉ－ｒｅｇへの値の取り込み元が演算値メモリＳｍｅｍに変わるだけで、前述のＡ×Ａの計算と同様である。

【０１２７】次に、ＲＥＤＣ関数を使用して、モンゴメリ手法を用いた剰余演算を実行する。

【０１２８】計算の方法は、Ａ×Ａの場合と同様であり、計算結果は、演算値メモリＡ’ｍｅｍとＷｌｍｅｍの双方に格納される。

【０１２９】べき乗剰余演算を行う場合は、式（２４）と式（２５）を繰り返し計算することになる。しかし、以上説明した各計算の実現方法を用いれば、式（２４）（２５）の計算結果は、ともに演算値メモリＡ’ｍｅｍとＷｌｍｅｍの双方に格納されるため、各々の式の計算の開始時に、演算値の初期設定等を行う必要がないので、繰り返し計算をスムーズに行うことができる。

【０１３０】〔Ａ・Ｒ’ｍｏｄＮ・・・（２６）の実現＊の方法〕Ａの値を演算値メモリＷｌｍｅｍに格納し、モンゴメリ法化におけるＮ’の値を演算値メモリＮ’ｍｅｍに格納し、Ｎの値を演算値メモリＮｍｅｍに格納する。

【０１３１】そして、タイミング／コントロール回路Ｔ／Ｃに対して、モード３信号を供給することにより、以下のように上記Ａ・Ｒ’ｍｏｄＮ・・・（２６）が実行される。

$$t = (X + m \cdot N) / R$$
$$= \{ (Wlmem) \times (Nmem) \text{ の結果のR以上の値} \} + 1 \cdots (30)$$

【０１３２】今回の式では、Ａ×？という乗算を実行する必要がない。従って、モンゴメリ手法を用いた剰余演算のみを実行する。

【０１３３】剰余演算におけるｍの計算の方法は、Ａ×Ａの場合とほぼ同様であり、計算結果は、Ａ’ｍｅｍ－Ｗｌｍｅｍの形で、演算値メモリＡ’ｍｅｍとＷｌｍｅｍの双方に格納したときの演算値メモリＷｌｍｅｍの値がｍの値となる。

【０１３４】また、ｔの計算は以下のようになる。

【０１３５】

式（２６）の計算では、上記のＸはＡであり、Ｒ以上の値はない（０と考えてよい）ので、式（２４）の実現方法で説明したようなＲ以上の値（Ｗｈｍｅｍ）の加算は行う必要がない。また（Ｘ＋ｍ・Ｎ）は、必ずＲの倍数になるのであるから、式（３０）の第２辺に示されるＸの値を加算する必要もなく、代わりに”〔（Ｗｌｍｅｍ）×（Ｎｍｅｍ）の結果のＲ以上の値〕＋１”で解が得られる。これは、暗号アルゴリズムの世界で剰余演算を使用するとき、本計算におけるＡの値が０になることがないからである。

【０１３６】従って、式（３０）では、まず（Ｗｌｍｅｍ）×（Ｎｍｅｍ）の計算を実行して、その結果をＡ’ｍｅｍ－Ｗｌｍｅｍの形で、演算値メモリＡ’ｍｅｍとＷｌｍｅｍに格納すればよい。その結果の得られたＡ’ｍｅｍ上の値が求める解となる。（本演算の実行でべき乗剰余演算のすべての演算は終了するため、ここでは演算値メモリＷｌｍｅｍへの格納を省略してもよい。）なお、このとき計算の最終結果を演算値メモリＡ’ｍｅｍの最下位桁に格納するタイミングで、図５に示した＋１端子に１を供給して、式（３０）の１を足す動作を実行する。

【０１３７】以上のように、第１の実施形態によれば、規定ビット長の乗加算器（乗算器と加算器を別個に設けてもよい）を演算部の中心として構成し、その周辺に設けられた回路に、タイミング／コントロール信号発生回路からの制御信号を供給することにより、モンゴメリ手法を用いた多ビット長の剰余演算あるいはべき乗剰余演算を実現できる。また、この方式によれば、演算部の中心が限られた（規定の）ビット長の演算器で構成できるため回路規模が小さくてすみ、ＬＳＩ化に適している。

【０１３８】（第２の実施形態）

（構成）図８は、本発明の第２の実施の形態の剰余演算コプロセッサを示す図である。

【０１３９】第２の実施形態は、第１の実施形態の構成（図５）に、演算値が０であることを検出して、次の演算のシーケンスを制御する回路を追加したものである。

【０１４０】具体的にこの制御回路は、図８における制御回路ＺｅｒｏＣである。この制御回路ＺｅｒｏＣは、演算値が０であることを検出して、次演算のシーケンスを制御する回路であり、高速加算器Ａｄｄの出力信号が与えられる入力端子と出力端子とを有している。この出力端子からは、タイミング／コントロール回路Ｔ／Ｃで発生するタイミング／コントロール信号及びコプロセッサ内の各種回路を動作させる信号に所定の制御を施す信号が出力する。

【０１４１】（動作）第１の実施形態の式（２４）（２５）の実現方法で述べた、ＲＥＤＣ関数の計算において、前の乗算の結果得られたＸの値は次の２つの場合分けができる。

【０１４２】（ａ）Ｒの倍数である場合・・前計算で得られた値（Ｗｌｍｅｍ）が０。

【０１４３】（ｂ）Ｒの倍数でない場合・・前計算で得られた値（Ｗｌｍｅｍ）が０でない。

【０１４４】本実施の形態では、制御回路ＺｅｒｏＣが、上記（ａ）（ｂ）のように、Ｘ値（Ａ×ＡやＡ×Ｂの結果の値）のＲ未満の値が０か否かを検出して、その結果に応じて、続く演算のシーケンスを次の通りに実行させる。

27

【０１４５】（ａ）の場合は、ＸｍｏｄＲ＝０なのでｍ＝０となるのは明かであるから、このときはｍの計算を行う必要はない。

＊【０１４６】そしてｔの計算は次のようになる。

【０１４７】

＊

$$t=(X+m\cdot N)/R=X/R=Whmem\cdots(31)$$

これは、Ａ×Ａ及びＡ×Ｂの計算結果の上位桁がそのままｔの値となっていることを示す。従って、この場合のハードウエア処理は、演算値メモリＷｈｍｅｍに格納された値をそのまま演算値メモリＡ’ｍｅｍとＷｌｍｅｍの双方に格納し、次の繰り返し演算に備える。すなわち、Ｘ値（前計算で得られた値）のＲ未満の値が０の場合は、ｍ、ｔの演算の必要がない。このことは、ｔの計算において、Ｘの全ビット長分を計算する必要がないことを意味する。

※【０１４８】（ｂ）の場合は、ｍを求めるための計算を第１の実施形態の通り実行し、計算結果をＡ’ｍｅｍ－Ｗｌｍｅｍの形で演算値メモリＡ’ｍｅｍとＷｌｍｅｍの双方に格納し、演算値メモリＷｌｍｅｍに格納された値をｍの値とする。そして、引き続きｔの計算を実行する。

【０１４９】そして、ｔの計算は次のようになる。

【０１５０】

※

$$t=(X+m\cdot N)/R$$
$$=Whmem+[(Wlmem)\times(Nmem)\text{の結果のR以上の値}]+1$$
$$\cdots(32)$$

式（３２）は、まず（Ｗｌｍｅｍ）×（Ｎｍｅｍ）の計算を実行して、その結果をＡ’ｍｅｍ－Ｗｌｍｅｍの形★

★で演算値メモリＡ’ｍｅｍとＷｌｍｅｍの双方に格納し、続いて

$$(A'mem)\times1+(Whmem)+1\cdots(33)$$

の計算を実行すれば解が得られる。なぜなら、Ｘ＋ｍ・Ｎの値は、必ずＲの倍数になるから、Ｒ未満の値をわざわざ計算対象にする必要がないからである。なお、式（３３）では、最下位桁の計算タイミングで、図５に示した＋１端子に１を供給して式（３３）の１を足す動作を実行する。

【０１５１】式（３３）で得られた計算結果は、演算値メモリＡ’ｍｅｍとＷｌｍｅｍの双方に格納するとともに、桁あふれが発生した場合は、キャリーフラグＣＦに１を書き込む。その後の処理は、第１の実施の形態と同様である。

【０１５２】以上のように、第２の実施形態によれば、演算値が０であることを検出して、次演算のシーケンスを制御する制御回路（乗加算器のビット長に相当する回路規模で済むので、小規模な回路である。）を付加することにより、第１の実施の形態で説明したコプロセッサの演算値メモリＭｍｅｍの削除と、演算量の削減が達成できる。従って、ハードウエアの削減と演算時間の短縮化を図ることができる。

【０１５３】（第３の実施形態）

（構成）図９は、本発明の第３の実施の形態の剰余演算コプロセッサを示す図である。

【０１５４】第３の実施形態は、第１の実施形態の構成（図５）もしくは第２の実施形態の構成（図８）に、オペランドビット長を選択してタイミング／コントロール信号を変化させるビット長選択制御回路を付加したものである。

【０１５５】具体的にこのビット長選択制御回路は、図９におけるＬｅｎＣｏｎｔである。このビット長選択制御回路ＬｅｎＣｏｎｔは、入力信号Ｓｅｌ－ｌｅｎに従って、タイミング／コントロール回路Ｔ／Ｃで発生する演算タイミング信号とコプロセッサ内の各種回路に供給するコントロール信号を制御するように動作する。

【０１５６】（動作）さて、本コプロセッサの動作は、オペランドビット長の変化に伴い、Ｒ値及びその値により決まるＲ’値、Ｎ’値のビット長と、規定ビット長乗加算の繰り返し手順（回数）とが変化するようになる。

【０１５７】例えば、高速乗加算器Ｍｕｌ／Ａｄｄがｌ６ビット長の時、オペランドビット長５１２ビットのＡ×Ａの乗算を実行する場合、高速乗加算器Ｍｕｌ／Ａｄｄによる乗加算の繰り返し回数は次のようになる。

【０１５８】

（５１２／１６）×（５１２／１６）＝１０２４　回

一方、オペランドビット長７６８ビットの場合は次のようになる。

【０１５９】

（７６８／１６）×（７６８／１６）＝２３０４　回

そして、これらの計算のシーケンスもそれぞれ異なる。また、前に述べたようにオペランドビット長によりＲ値は一義的に決まるので、Ｒ’値、Ｎ’値のビット長もそれに応じて変化する。

【０１６０】ビット長選択制御回路ＬｅｎＣｏｎｔは、これらの変化要因を制御する、すなわち、選択されたビット長に対応して、タイミング／コントロール回路Ｔ／Ｃで発生させる演算タイミング信号やコントロール信号が出力するよう制御する。

【０１６１】一般的に、ビット長選択制御回路ＬｅｎＣｏｎｔは、ＰＬＡや論理回路等の比較的小規模な回路構成で実現できる。

【０１６２】以上のように、第３の実施形態のコプロセッサでは、ビット長選択制御回路ＬｅｎＣｏｎｔを付加することによって、さまざまなオペランドビット長の剰

余演算あるいはべき乗剰余演算を実行することが可能となる。

【０１６３】（第４の実施形態）

（構成）以上の実施例で示したコプロセッサの演算の基本は、規定ビット長の乗加算である。この基本乗加算を繰り返して剰余演算あるいはべき乗剰余演算を実現する方法は、以上の実施例で示した通りである。

【０１６４】しかしながら、以上の実施例では、剰余演算モードのみを行うコプロセッサであったため、演算の基本が乗加算であるにも関わらず、その用途が剰余演算に限られている。

【０１６５】そこで、さまざまな演算の基本となる多ビット長乗加算のモードを追加して、本コプロセッサの汎用性を向上させる。

【０１６６】図１０は、本発明の第４の実施の形態の剰余演算コプロセッサを示す図である。

【０１６７】第４の実施の形態は、先の実施形態に乗加算のモードを追加したものであり、このモードを実行するためにタイミング／コントロール回路Ｔ／Ｃにモード信号４を供給するようにしたものである。

【０１６８】（動作）以下に示す乗加算を実行する例を以下に説明する。

【０１６９】Ａ・Ｂ＋Ｃ・・・（３４）

まず、式（３４）において、値Ａを演算値メモリＷｌｍｅｍに、値Ｂを演算値メモリＳｍｅｍに、値Ｃを演算値メモリＷｈｍｅｍに格納する。そして、タイミング／コントロール回路Ｔ／Ｃにモード信号４を供給して、演算の種類を、乗加算を実行するモードにする。

【０１７０】上述の動作をハードウエアイメージでより詳細に表現すると図１１のようになる。

【０１７１】この演算は、演算値メモリＳｍｅｍの各アドレスＳ３、Ｓ２、Ｓ１、Ｓ０に格納された値と演算値メモリＷｌｍｅｍの各アドレスＷｌ３、Ｗｌ２、Ｗｌ１、Ｗｌ０に格納された値とを乗算し、この結果と、演算値メモリＷｈｍｅｍの各アドレスＷｈ３、Ｗｈ２、Ｗｈ１、Ｗｈ０に格納された値とを足し合わせたときの値が、演算値メモリＷｈｅｍ及び演算値メモリＷｌｍｅｍの各アドレスＷｈ３、Ｗｈ２、Ｗｈ１、Ｗｈ０、Ｗｌ３、Ｗｌ２、Ｗｌ１、Ｗｌ０に格納されることを意味する。

【０１７２】（単位乗加算１）まず、レジスタＹｉ－ｒｅｇが、演算値メモリＷｌｍｅｍのアドレスＷｌ０に格納された値（被乗算値）を取り込むとともに、レジスタＸｉ－ｒｅｇが演算値メモリＳｍｅｍのアドレスＳ０に格納された値（乗算値）を取り込み、レジスタＡｉ－ｒｅｇが演算値メモリＷｈｍｅｍのアドレスＷｈ０に格納された値（加算値）を取り込む。次に、高速乗加算器Ｍｕｌ／Ａｄｄはこの乗算値と被乗算値を乗算し、かつこの乗算結果に加算値を加算し高速加算器Ａｄｄに向けて出力する。この単位乗加算１では、高速加算器ＡｄｄのＴｈｒｏｕｇｈ端子に”ｈｉｇｈレベル”を供給して、レジスタＲＨの内容を加算しないようにする。

【０１７３】そして、この単位乗加算における演算結果の下位桁は、本乗加算の最終演算結果の最下位桁であるので、この単位乗加算１の演算結果の下位桁は本乗算の最終演算結果として演算値メモリＷｌｍｅｍの最下位アドレスＷｌ０に格納される。（図１１において、アンダーラインが付与されたアドレスには、最終演算結果が格納されていることを意味する。）この単位乗加算１の演算結果の上位桁は、次の単位乗加算のためにレジスタＲＨに格納される。

【０１７４】（単位乗加算２）レジスタＹｉ－ｒｅｇは、すでに格納された値（被乗算値）を維持し、レジスタＸｉ－ｒｅｇは、演算値メモリＳｍｅｍのアドレスＳ１に格納された値（乗算値）を取り込み、レジスタＡｉ－ｒｅｇが演算値メモリＷｈｍｅｍのアドレスＷｈ１に格納された値（加算値）を取り込む。次に、高速乗加算器Ｍｕｌ／Ａｄｄはこの乗算値と被乗算値を乗算し、かつこの乗算結果に加算値を加算し、高速加算器Ａｄｄに向けて出力する。この単位乗加算２では、高速加算器ＡｄｄのＴｈｒｏｕｇｈ端子に”ｌｏｗレベル”を供給して、レジスタＲＨの内容（単位乗加算１の演算結果の上位桁）を高速乗加算器Ｍｕｌ／Ａｄｄの出力に加算する。この単位乗加算２の演算結果の下位桁は、演算値メモリＷｈｍｅｍの最下位アドレスＷｈ０に格納される。この単位乗加算２の演算結果の上位桁は、次の単位乗加算の桁合わせのためにレジスタＲＨに格納される。

【０１７５】（単位乗加算３）レジスタＹｉ－ｒｅｇは、すでに格納された値（被乗算値）を維持し、レジスタＸｉ－ｒｅｇは、演算値メモリＳｍｅｍのアドレスＳ２に格納された値（乗算値）を取り込み、レジスタＡｉ－ｒｅｇが演算値メモリＷｈｍｅｍのアドレスＷｈ２に格納された値（加算値）を取り込む。次に、高速乗加算器Ｍｕｌ／Ａｄｄはこの乗算値と被乗算値を乗算し、かつこの乗算結果に加算値を加算し、高速加算器Ａｄｄに向けて出力する。この単位乗加算３では、高速加算器ＡｄｄのＴｈｒｏｕｇｈ端子に”ｌｏｗレベル”を供給して、レジスタＲＨの内容（単位乗加算２の演算結果の上位桁）を高速乗加算器Ｍｕｌ／Ａｄｄの出力に加算する。この単位乗加算３の演算結果の下位桁は、演算値メモリＷｈｍｅｍの最下位アドレスＷｈ１に格納される。この単位乗加算３の演算結果の上位桁は、次の単位乗加算の桁合わせのためにレジスタＲＨに格納される。

【０１７６】（単位乗加算４）レジスタＹｉ－ｒｅｇは、すでに格納された値（被乗算値）を維持し、レジスタＸｉ－ｒｅｇは、演算値メモリＳｍｅｍのアドレスＳ３に格納された値（乗算値）を取り込み、レジスタＡｉ－ｒｅｇが演算値メモリＷｈｍｅｍのアドレスＷｈ３に格納された値（加算値）を取り込む。次に、高速乗加算器Ｍｕｌ／Ａｄｄはこの乗算値と被乗算値を乗算し、か

つこの乗算結果に加算値を加算し、高速加算器Ａｄｄに向けて出力する。この単位乗加算４では、高速加算器ＡｄｄのＴｈｒｏｕｇｈ端子に”ｌｏｗレベル”を供給して、レジスタＲＨの内容（単位乗加算３の演算結果の上位桁）を高速乗加算器Ｍｕｌ／Ａｄｄの出力に加算する。この単位乗加算４の演算結果の下位桁は、演算値メモリＷｈｍｅｍの最下位アドレスＷｈ２に格納される。この単位乗加算４の演算結果の上位桁は、次の単位乗加算の桁合わせのために演算値メモリＷｈｍｅｍのアドレスＷｈ３に格納される。

【０１７７】（単位乗加算５）レジスタＹｉ－ｒｅｇは、演算値メモリＷｌｍｅｍのアドレスＷｌｌに格納された値（被乗算値）を取り込み、レジスタＸｉ－ｒｅｇは、演算値メモリＳｍｅｍのアドレスＳ０に格納された値（乗算値）を取り込み、レジスタＡｉ－ｒｅｇは、演算値メモリＷｈｍｅｍのアドレスＷｈ０に格納された値（加算値）を取り込む。次に、高速乗加算器Ｍｕｌ／Ａｄｄはこの乗算値と被乗算値を乗算し、その乗算結果に加算値を加算し、高速加算器Ａｄｄに向けて出力する。この単位乗加算５では、高速加算器ＡｄｄのＴｈｒｏｕｇｈ端子に”ｈｉｇｈレベル”を供給して、レジスタＲＨの内容（単位乗加算４の演算結果の上位桁）を加算しないようにする。この単位乗加算５の演算結果の下位桁は、演算値メモリＷｌｍｅｍのアドレスＷｌｌに格納される。この単位乗加算５の演算結果の上位桁は、次の単位乗加算の桁合わせのためにレジスタＲＨに格納される。

【０１７８】（単位乗加算６）レジスタＹｉ－ｒｅｇは、すでに格納された値（被乗算値）を維持し、レジスタＸｉ－ｒｅｇは、演算値メモリＳｍｅｍのアドレスＳ１に格納された値（乗算値）を取り込み、レジスタＡｉ－ｒｅｇは、演算値メモリＷｈｍｅｍのアドレスＷｈ１に格納された値（加算値）を取り込む。次に、高速乗加算器Ｍｕｌ／Ａｄｄはこの乗算値と被乗算値を乗算し、その乗算結果に加算値を加算し、高速加算器Ａｄｄに向けて出力する。この単位乗加算６では、高速加算器ＡｄｄのＴｈｒｏｕｇｈ端子に”ｌｏｗレベル”を供給して、レジスタＲＨの内容（単位乗加算５の演算結果の上位桁）を高速乗加算器Ｍｕｌ／Ａｄｄの出力に加算する。この単位乗加算６の演算結果の下位桁は、演算値メモリＷｈｍｅｍのアドレスＷｈ０に格納される。この単位乗加算６の演算結果の上位桁は、次の単位乗加算の桁合わせのためにレジスタＲＨに格納される。

【０１７９】（単位乗加算７）レジスタＹｉ－ｒｅｇは、すでに格納された値（被乗算値）を維持し、レジスタＸｉ－ｒｅｇは、演算値メモリＳｍｅｍのアドレスＳ２に格納された値（乗算値）を取り込み、レジスタＡｉ－ｒｅｇは、演算値メモリＷｈｍｅｍのアドレスＷｈ２に格納された値（加算値）を取り込む。次に、高速乗加算器Ｍｕｌ／Ａｄｄはこの乗算値と被乗算値を乗算し、その乗算結果に加算値を加算し、高速加算器Ａｄｄに向けて出力する。この単位乗加算７では、高速加算器ＡｄｄのＴｈｒｏｕｇｈ端子に”ｌｏｗレベル”を供給して、レジスタＲＨの内容（単位乗加算６の演算結果の上位桁）を高速乗加算器Ｍｕｌ／Ａｄｄの出力に加算する。この単位乗加算７の演算結果の下位桁は、演算値メモリＷｈｍｅｍのアドレスＷｈ１に格納される。この単位乗加算７の演算結果の上位桁は、次の単位乗加算の桁合わせのためにレジスタＲＨに格納される。

【０１８０】（単位乗加算８）レジスタＹｉ－ｒｅｇは、すでに格納された値（被乗算値）を維持し、レジスタＸｉ－ｒｅｇは、演算値メモリＳｍｅｍのアドレスＳ３に格納された値（乗算値）を取り込み、レジスタＡｉ－ｒｅｇは、演算値メモリＷｈｍｅｍのアドレスＷｈ３に格納された値（加算値）を取り込む。次に、高速乗加算器Ｍｕｌ／Ａｄｄはこの乗算値と被乗算値を乗算し、その乗算結果に加算値を加算し、高速加算器Ａｄｄに向けて出力する。この単位乗加算８では、高速加算器ＡｄｄのＴｈｒｏｕｇｈ端子に”ｌｏｗレベル”を供給して、レジスタＲＨの内容（単位乗加算７の演算結果の上位桁）を高速乗加算器Ｍｕｌ／Ａｄｄの出力に加算する。この単位乗加算８の演算結果の下位桁は、演算値メモリＷｈｍｅｍのアドレスＷｈ２に格納される。この単位乗加算８の演算結果の上位桁は、演算値メモリＷｈｍｅｍのアドレスＷｈ３に格納される。

【０１８１】単位乗加算９から１６までは、前述の単位乗加算に習って、図１１のごとく実行される。そして、最終演算結果は、Ｗｈｍｅｍ－Ｗｌｍｅｍ（”－”は、引き算を示すものではない。）の形で格納される。

【０１８２】図１２には、８５９１×４６７３＋２０６９＝４０１４７８１２を、上述のハードウエアで実行した場合の単位乗加算を示す。動作の説明については、図１１と同様であるので省略する。以上のように、第４の実施形態によれば、さまざまな演算の基本となる多ビット長の乗加算のモードを実現できるので、コプロセッサの汎用性を大幅に向上させることができる。

【０１８３】（第５の実施形態）

（構成）以上の実施の形態で示したコプロセッサは、高速乗加算器Ｍｕｌ／Ａｄｄと高速加算器Ａｄｄの入力ビット長で決まる単位乗加算毎に、メモリへのアクセスを行う必要があるため、演算時間の短縮化においては改善の余地がある。なぜなら、メモリを含んだ回路全体の動作速度は、メモリアクセスタイムが回路全体の動作速度を制限してしまうからである。

【０１８４】そこで、本実施の形態では、単位乗加算の演算回数に対するメモリのアクセス回数を減らして、最終的な演算時間の短縮を図る。

【０１８５】図１３は、本発明の第５の実施の形態の剰余演算コプロセッサを示す図である。

【０１８６】第５の実施の形態は、先の実施の形態の構

成に対して、被乗算値格納レジスタＹｉ－ｒｅｇを複数個に増やして、その出力を選択する回路を設けたものである。さらに、高速加算器Ａｄｄの上位桁あるいは下位桁の出力値を格納するレジスタも被乗算値格納レジスタの個数に対応する個数設け、各レジスタの出力値を適当に選択する選択回路を設けたものである。

【０１８７】具体的には、レジスタＹｉ－ｒｅｇ［０］、レジスタＹｉ－ｒｅｇ［１］、レジスタＹｉ－ｒｅｇ［２］、レジスタＹｉ－ｒｅｇ［３］が、高速乗加算器Ｍｕｌ／Ａｄｄの入力ビット長に対応した長さを持つ被乗算値レジスタであり、各々の出力は、被乗算値選択回路Ｙｉ－ｓｅｌに接続されている。選択回路Ｙｉ－ｓｅｌは、レジスタＹｉ－ｒｅｇ［０］、レジスタＹｉ－ｒｅｇ［１］、レジスタＹｉ－ｒｅｇ［２］、レジスタＹｉ－ｒｅｇ［３］の値のうちのいずれか１つを選択して、後続の高速乗加算器Ｍｕｌ／Ａｄｄに与える選択回路である。

【０１８８】レジスタＲＡは、高速加算器Ａｄｄの出力のうち上位桁Ｒ－ｈｉｇｈを一時格納するレジスタであり、その出力は後続の選択回路ＳｅｌＡ、ＳｅｌＢ、ＥｎＳｅｌに接続されている。そして高速加算器Ａｄｄ出力のうち下位桁Ｒ－ｌｏｗは、選択回路ＳｅｌＢ及びイネーブルバッファＥｎに接続されている。

【０１８９】レジスタＡｉ－ｒｅｇの出力は、選択回路ＳｅｌＡに接続されている。そして選択回路ＳｅｌＡは、レジスタＡｉ－ｒｅｇとレジスタＲＡの内容のうちいずれかを選択して高速加算器Ａｄｄに加算入力すべく加算加算器Ａｄｄに接続されている。

【０１９０】選択回路ＳｅｌＢは、Ｒ－ｌｏｗとレジスタＲＡの内容のうちいずれかを選択してレジスタＲＢに向けてその結果を出力すべくレジスタＲＢに接続されている。

【０１９１】レジスタＲＢは、選択回路ＳｅｌＢの出力を一時格納するレジスタであり、出力はレジスタＲＣ及びＥｎｓｅｌに接続されている。

【０１９２】レジスタＲＣは、レジスタＲＢの出力を一時格納するレジスタであり、出力はレジスタＲＤ及びＥｎｓｅｌに接続されている。

【０１９３】レジスタＲＤは、レジスタＲＣの出力を一時格納するレジスタであり、出力は選択回路Ｅｎｓｅｌ及び高速乗加算器Ｍｕｌ／Ａｄｄに接続されている。

【０１９４】選択回路Ｅｎｓｅｌは、レジスタＲＡ、ＲＢ、ＲＣ、ＲＤの内容のうちいずれか１つを選択して演算値メモリに向けて出力する選択回路である。

【０１９５】その他の回路については、先の実施の形態と同様な働きをするので説明を省略する。

【０１９６】（動作）次に、演算の基本となる
（Ａ３Ａ２Ａ１Ａ０）×Ｓ０＋Ｃ０
の計算方法について、
（Ｗ１３、Ｗ１２、Ｗ１１、Ｗ１０）×Ｓ０＝（ＲＡ、ＲＢ、ＲＣ、ＲＤ、Ｗ１０）
のようなハードウエアイメージを使用して、図１４を用いて説明する。

【０１９７】なお、アンダーラインが付与されたアドレスは、演算の最終結果が格納されていることを意味する。

【０１９８】（初期設定）まず図示のごとく初期設定が行われる。

【０１９９】（ステップ１）レジスタＸｉ－ｒｅｇの値と、選択回路Ｙｉ－ｓｅｌで選択されたレジスタＹｉ－ｒｅｇ［０］の値と、レジスタＲＤの値と選択回路ＳｅｌＡで選択されたレジスタＡｉ－ｒｅｇの値との間で演算が行われる。そして、この演算の終了間際には、演算結果の上位桁であるＲ－ｈｉｇｈは、レジスタＲＡに、演算結果の下位桁であるＲ－ｌｏｗは、演算値メモリＷｌｍｅｍに格納される。同時に、レジスタＲＣの値はレジスタＲＤに、レジスタＲＢの値はレジスタＲＣに、レジスタＲＡの値は選択回路ＳｅｌＢで選択されたレジスタＲＢに格納される。

（ステップ２）レジスタＸｉ－ｒｅｇの値と、選択回路Ｙｉ－ｓｅｌで選択されたレジスタＹｉ－ｒｅｇ［１］の値と、レジスタＲＤの値と選択回路ＳｅｌＡで選択されたレジスタＲＡの値との間で演算が行われる。そして、この演算の終了間際には、演算結果の上位桁であるＲ－ｈｉｇｈは、レジスタＲＡに、演算結果の下位桁であるＲ－ｌｏｗは、選択回路ＳｅｌＢで選択されたレジスタＲＢに格納される。同時に、レジスタＲＣの値はレジスタＲＤに、レジスタＲＢの値はレジスタＲＣに格納される。

【０２００】（ステップ３）レジスタＸｉ－ｒｅｇの値と、選択回路Ｙｉ－ｓｅｌで選択されたレジスタＹｉ－ｒｅｇ［２］の値と、レジスタＲＤの値と選択回路ＳｅｌＡで選択されたレジスタＲＡの値との間で演算が行われる。そして、この演算の終了間際には、演算結果の上位桁であるＲ－ｈｉｇｈは、レジスタＲＡに、演算結果の下位桁であるＲ－ｌｏｗは、選択回路ＳｅｌＢで選択されたレジスタＲＢに格納される。同時に、レジスタＲＣの値はレジスタＲＤに、レジスタＲＢの値はレジスタＲＣに格納される。

【０２０１】（ステップ４）レジスタＸｉ－ｒｅｇの値と、選択回路Ｙｉ－ｓｅｌで選択されたレジスタＹｉ－ｒｅｇ［３］の値と、レジスタＲＤの値と選択回路ＳｅｌＡで選択されたレジスタＲＡの値との間で演算が行われる。そして、この演算の終了間際には、演算結果の上位桁であるＲ－ｈｉｇｈは、レジスタＲＡに、演算結果の下位桁であるＲ－ｌｏｗは、選択回路ＳｅｌＢで選択されたレジスタＲＢに格納される。同時に、レジスタＲＣの値はレジスタＲＤに、レジスタＲＢの値はレジスタＲＣに格納される。

【０２０２】以上のステップ１からステップ４までの処

理では、演算値メモリへのアクセスは、ステップ１で１度行っているだけである。従って、残りのステップ２からステップ４までの期間に、各演算値メモリのアドレスの変更、プリチャージ等を行ってメモリアクセスの時間をかせぐことができる。また、ステップ２からステップ４までの期間内に行う動作は、ほとんど同じであるため、演算処理が複雑になることはない。

【０２０３】次に、他の演算の例として、

（Ａ３、Ａ２、Ａ１、Ａ０）×（Ｂ３、Ｂ２、Ｂ１、Ｂ０）＋（Ｃ３、Ｃ２、Ｃ１、Ｃ０）

の計算方法について、

（Ｗ１３、Ｗ１２、Ｗ１１、Ｗ１０）×（Ｓ３、Ｓ２、Ｓ１、Ｓ０）＋（Ｗｈ３、Ｗｈ２、Ｗｈ１、Ｗｈ０）＝（ＲＡ、ＲＢ、ＲＣ、ＲＤ、Ｗ１３、Ｗ１２、Ｗ１１、Ｗ１０）

のようなハードウエアイメージを使用して、図１５、図１６に示す。

【０２０４】さらに、実際の計算の例として、図１７に８５９１×４６７３＋２０６９＝４０１４７８１２のタイム１からタイム４までの演算過程を示す。

【０２０５】なお、アンダーラインが付与された値は、演算の最終結果であることを意味する。

【０２０６】この図から、最終結果の上位の桁は、タイム４のステップ４で得られたレジスタＲＡ、ＲＢ、ＲＣ、ＲＤに格納された値であり、下位の桁はステップ１で得られた演算値メモリＷｌｍｅｍの値であることが理解できる。

【０２０７】以上のように第５の実施形態によれば、乗加算器等の演算処理部のサイズを変える必要はなく、全体的に小規模な回路で演算時間の短い剰余演算コプロセッサを実現できる。

【０２０８】（第６の実施形態）

（構成）図１８は、本発明の第６の実施の形態を示すブロック図であり、先に説明したコプロセッサと外部装置との間に、演算値メモリインターフェイス回路と演算コントロール回路を設けたものである。

【０２０９】（演算値メモリインタフェイス回路ＭｅｍＩＦ）ＭｅｍＩＦは、ＭＣＵとコプロセッサ内の演算値メモリとの間でデータの授受を行うための演算値メモリインタフェイス回路である。

【０２１０】演算値メモリは、演算に先だってコプロセッサ外部から演算データを格納し、演算の終了時に演算結果の値をコプロセッサ外部に送出する。同時に、演算実行中はコプロセッサ外部とは関係なくダイナミックに演算ユニットとのアクセスを繰り返す。つまり、演算値メモリは２種類の通信プロトコルを有する。演算値メモリインタフェイスＭｅｍＩＦでは、それを実現すべく回路を構成する。

【０２１１】ＭＣＵから出力されるアドレス信号ａｄｒｓとメモリ制御信号Ｍｅｍｃｏｎは、演算値メモリインタフェイスＭｅｍＩＦに入力され、演算値メモリインタフェイスＭｅｍＩＦ内で、コプロセッサ内の各演算値メモリ別のアドレス信号Ｃｏｍｅｍａｄとメモリコントロール信号Ｃｏｍｃｏｎを作成してコプロセッサに供給する。ＭＤｂｕｓは、ＭＣＵのデータバスであり、ＣｏＤｂｕｓはコプロセッサの外部インタフェイス用データバスである。

【０２１２】ここで、各演算値メモリがＭＣＵ内のある単一のメモリ空間上に配置されている場合は、演算値メモリインタフェイスＭｅｍＩＦに入力されるａｄｒｓ信号とＭｅｍｃｏｎ信号とＭＤｂｕｓは一種類でよいが、複数のメモリ空間上に配置されている場合は複数種類の入力となる。また、一般的にはＭＤｂｕｓとＣｏＤｂｕｓは直結されることが多いが、例えばコプロセッサ内部で処理される演算値メモリのデータ長とＭＣＵのデータ長が異なる場合などは、演算値メモリインタフェイスＭｅｍＩＦを介してデータ変換を行うこともできる。

【０２１３】なお、コプロセッサが演算実行中は、演算値メモリが演算実行のためにダイナミックに動作しているので、ＭＣＵからのアクセスを禁止するように制御されている。このようにして、コプロセッサ内演算値メモリと外部装置との第１の通信プロトコルを実現する。

【０２１４】次に、演算実行の際、演算値メモリインタフェイスＭｅｍＩＦにはコプロセッサ内のタイミング／コントロール回路Ｔ／Ｃから出力される演算実行時メモリコントロール信号Ｅｘｅｍｃが入力され、それを受け取った演算値メモリインタフェイスＭｅｍＩＦは、演算値メモリが演算ユニットとの間でデータの授受を行えるように処理されたＣｏｍｅｍａｄ信号とＣｏｍｃｏｎ信号をコプロセッサに向けて出力する。このようにして、コプロセッサ内の演算実行中の演算値メモリと演算ユニットとの第２の通信プロトコルを実現する。

【０２１５】（演算コントロール回路ＣｏｐＣｏｎ）ＣｏｐＣｏｎは、ＭＣＵから出力されるコプロセッサ制御信号Ｅｘｃｏｎを受け取って、コプロセッサ内部へ演算制御信号Ｓｅｖｅｘ（以上の実施の形態で言えば、演算モード信号、ビット長選択信号等）を供給する演算コントロール回路である。ここで、コプロセッサへの演算の開始を促すのは、演算モード信号やコプロセッサ原振クロックＣＯＰＣＬＫを供給することで実現できる。

【０２１６】また、演算の終了をＭＣＵ側で確認するためには、コプロセッサ内のタイミング／コントロール回路Ｔ／Ｃで発生される演算終了のタイミング信号ＣｏｅｎｄをＣｏｐＣｏｎで受取り、ラッチ回路などで処理した演算終了モニター信号ＥｎｄｍｏｎｉをＭＣＵに向けて出力する。

【０２１７】一般に、演算コントロール回路ＣｏｐＣｏｎは、ＭＣＵのペリフェラル回路としてローカルメモリエリア等に割り付けてＭＣＵの命令で直接アクセスできるように構成でき、比較的に簡単な回路で実現できる。

【０２１８】なお、図中のＭＣＵＣＬＫ信号は、ＭＣＵの原振クロックを、ＣＯＰＣＬＫ信号はコプロセッサの原振クロックを示している。

【０２１９】以上のように第６の実施の形態によれば、比較的に小規模な回路で外部装置（例えば、ＭＣＵ等）とコプロセッサとのインタフェイスが実現できるので、外部インタフェイス付きの剰余演算コプロセッサ、あるいは剰余演算コプロセッサ内蔵のＭＣＵを構成でき、しかもＬＳＩ化が可能となる。

【０２２０】（第７の実施形態）第６の実施の形態では、演算終了をコプロセッサ外部で確認する手段は、演算終了モニター信号Ｅｎｄｍｏｎｉを見るしかない。しかしこの方法では、コプロセッサで処理される多ビット長の剰余演算量は多いため、例えば外部装置がＭＣＵの場合はＭＣＵ側で常にＥｎｄｍｏｎｉ信号を監視する時間が比較的長くなり、ＭＣＵの動作パフォーマンスが低下する。

【０２２１】本実施の形態はこの問題を解決するために、専用の演算終了時の割り込み制御回路を設けたものである。

【０２２２】図１９は、本発明の第７の実施の形態を示すブロック図である。

【０２２３】図１９中で、ＩｎｔＣｏｎは演算終了割り込み制御回路であり、あらかじめＭＣＵから出力される割り込み設定信号Ｉｎｔｓｅｔにより、割り込みの準備がなされている。コプロセッサより演算終了のタイミング信号Ｃｏｅｎｄが入力されると、ＣｏｐＣｏｎで設定された演算モードの種類別に、割り込みに必要な割り込み処理リクエスト信号、アクノリッジ信号、ベクター制御信号等をＩｎｔｓｉｇとして、ＭＣＵとの間でやりとりし、最後に割り込みの準備を解除して割り込み処理を終了する。

【０２２４】なお、演算終了の割り込み要因は１つに固定しても良いが、本コプロセッサは複数の演算モードを有しているため、各演算モード毎の割り込みを行った方が外部装置によるべき乗剰余演算への展開の仕方が容易になるので、割り込み要因を複数に設定したほうが良い。

【０２２５】Ｉｎｔｃｏｎは、演算コントロール回路ＣｏｐＣｏｎと同様にＭＣＵのペリフェラル回路としてローカルメモリエリア等に割り付けてＭＣＵの命令で直接アクセスできるように構成でき、比較的に簡単な回路で実現できる。

【０２２６】以上のように、第７の実施の形態によれば、比較的に小規模な回路で、演算終了割り込み機能を持った、外部インタフェイス付きの剰余演算コプロセッサ、あるいは剰余演算コプロセッサ内蔵のＭＣＵを構成でき、しかもＬＳＩ化が可能となる。

【０２２７】（第８の実施形態）本コプロセッサ内での剰余演算は、演算開始から終了までダイナミックに実行されるため、外部装置と接続した際のシステム全体の演算実行中の消費電流を抑えるために、演算実行時の外部装置の動作を一時停止（以下、スリープと呼ぶ。）状態にすると都合がよい。

【０２２８】本提案は、それを実現するために、専用の外部装置スリープ制御回路とクロック制御回路を設けたものである。

【０２２９】図２０は、第８の実施形態を示す図であり、第７の実施形態に、外部装置スリープ制御回路とクロック制御回路を付加したものである。

【０２３０】図２０中で、ＳｌｐＣｏｎはＭＣＵスリープ制御回路であり、ＭＣＵからのスリープ設定信号Ｓｌｐｓｅｔを受けてスリープ信号Ｓｌｐを発生させ、その信号をクロック制御回路ＣＬＫＣｏｎに送る。

【０２３１】ＣＬＫＣｏｎは、通常は外部より入力されたシステムクロックＣＬＫ信号をもとに、ＭＣＵに対してＭＣＵ原振クロックＭＣＵＣＬＫを供給しているが、いったんＳｌｐ信号を受け取るとＭＣＵへのＭＣＵＣＬＫの供給をやめるように働く。

【０２３２】演算終了時には、演算終了のタイミング信号ＣｏｅｎｄがＳｌｐＣｏｎに入力されてＳｌｐ信号のＣＬＫＣｏｎへの供給がなくなり、ＣＬＫＣｏｎからＭＣＵに対してＭＣＵＣＬＫの供給を再開する。

【０２３３】なお、このスリープ機能は一般的には外部装置のある端子への信号入力等でスリープ解除することが多く、この場合もそれを併用するようにＳｌｐＣｏｎを構成しても良い。

【０２３４】ＳｌｐＣｏｎは、演算コントロール回路ＣｏｐＣｏｎと同様にＭＣＵのペリフェラル回路として、ローカルメモリエリア等に割り付けてＭＣＵの命令で直接アクセスできるように構成でき、比較的に簡単な回路で実現できる。

【０２３５】以上のように、第８の実施形態によれば、比較的に小規模な回路で、外部装置スリープ機能を持った、外部インターフェイス付きの剰余演算コプロセッサ、あるいは剰余演算コプロセッサ内蔵のＭＣＵを構成でき、システム全体の低消費電力化が実現して、しかもＬＳＩ化が可能となる。

【０２３６】（第９の実施形態）本コプロセッサの用途は、大規模で複雑な剰余演算を行う暗号アルゴリズム等の、処理時間の高速性を問われる場合に使用することができる。従って、システムに供給される入力原振クロックの周波数がいかなる場合であっても、演算時間は短いことが要求される。

【０２３７】そこで、逓倍速クロック制御回路を内蔵して演算速度の向上機能を付加する。

【０２３８】図２１は、第９の実施の形態を示す図であり、第８の実施の形態のクロック制御回路をモディファイして逓倍速クロック制御回路を構成した場合の図である。なお、第６、第７の実施の形態に逓倍速クロック制

御回路を別個に付加した場合でも構わない。

【０２３９】図２１中、ＣＬＫＣｏｎ２は、逓倍速クロック制御回路を内蔵したクロック制御回路であり、ＭＣＵから逓倍速設定用の信号としてＣｋｗｓｅｔが入力されると逓倍速機能が働く。

【０２４０】逓倍速機能は、外部から供給される原振クロックＣＬＫに対して動作し、作成した逓倍速クロックがＭＣＵの原振クロックＭＣＵＣＬＫ、あるいはコプロセッサの原振クロックＣＯＰＣＬＫ、あるいはその両方に出力させるように回路を構成しておく。そうすることで、ユーザーにとっては、消費電流との兼ね合いから、システムの上でバリエーションに富んだ選択ができるようになる。

【０２４１】逓倍速機能の解除も、設定の場合と同様に解除信号としてのＣｋｗｓｅｔ信号を入力することによって、実現する。

【０２４２】逓倍速クロック制御回路は、演算コントロールＣｏｐＣｏｎと同様にＭＣＵのペリフェラル回路としてローカルメモリエリア等に割り付けてＭＣＵの命令で直接アクセスできるように構成でき、比較的に簡単な回路で実現できる。

【０２４３】以上のように、第９の実施の形態によれば、比較的に小規模な回路で、システムの逓倍速機能を持った、外部インタフェイス付きの剰余演算コプロセッサ、あるいは剰余演算コプロセッサ内臓のＭＣＵを構成でき、システム全体の演算処理速度が向上して、しかもＬＳＩ化が可能となる。

【０２４４】第１から第５の実施の形態で示した高速乗加算器Ｍｕｌ／Ａｄｄは、乗算器と加算器に分離した構成としても良い。

【０２４５】第１から第５の実施の形態で示した高速乗加算器Ｍｕｌ／Ａｄｄ、高速加算器Ａｄｄは、一体化されたものであっても良い。

【０２４６】第１から第５の実施の形態で示した高速乗加算器Ｍｕｌ／Ａｄｄ、高速加算器Ａｄｄは、市販のＩＣで構成しても良い。

【０２４７】第６から第９の実施の形態で示したコプロセッサは、第１から第５の実施の形態で示したコプロセッサをもとにした記述をしているが、そのコプロセッサと同等の機能を持つ他のコプロセッサで構成しても良い。

【０２４８】第１から第５の実施の形態で示した多ビット長乗算アルゴリズムは、一般的なもので説明しているが、説明の中で使用しているハードウエア構成で使用できる他の乗算アルゴリズム（例えば、ＢＯＯＴＨ等）をタイミング／コントロール回路で作り出して実現しても良い。

【０２４９】第２から第５の実施の形態の説明では、”０”検出回路ＺｅｒｏＣの使用例を演算値メモリの削減を主な目的として記述しているが、ＺｅｒｏＣは例えば単位乗算の乗算値あるいは被乗算値の内容が０であった場合に、演算動作をオミットし演算結果の途中値を無条件に０として次の演算に進むようにシーケンスを制御して、演算時間を短縮する目的にも使用できる。

【０２５０】第５の実施の形態では、被乗算値格納レジスタの個数を増やす記述をしているが、被乗算値格納レジスタのビット長を大きくしてサイズを増やす場合も可能である。

【０２５１】

【発明の効果】以上詳細に説明したように、本発明の代表的なものによれば、べき乗剰余演算の解を求めるに際し、予め用意されたモード信号を供給するのみで、各種演算を実行することができる。

【図面の簡単な説明】

【図１】本発明におけるフローチャートを示す図である。

【図２】暗号文の処理の具体例を説明する図である。

【図３】べき乗剰余演算の概略フローである。

【図４】本発明のハードウエアの概略を示すブロック図である。

【図５】本発明の第１の実施の形態を示す図である。

【図６】単位乗加算を説明する図である。

【図７】本発明の第１の実施形態の動作を説明するハードウエアイメージである。

【図８】本発明の第２の実施の形態を示す図である。

【図９】本発明の第３の実施の形態を示す図である。

【図１０】本発明の第４の実施の形態を示す図である。

【図１１】本発明の第４の実施形態の動作を説明するハードウエアイメージである。

【図１２】本発明の第４の実施形態の動作を説明する演算例である。

【図１３】本発明の第５の実施の形態を示す図である。

【図１４】本発明の第５の実施形態の動作を説明するハードウエアイメージである。

【図１５】本発明の第５の実施形態の動作を説明するハードウエアイメージである。

【図１６】本発明の第５の実施形態の動作を説明するハードウエアイメージである。

【図１７】本発明の第５の実施形態の動作を説明する演算例である。

【図１８】本発明の第６の実施の形態を示す図である。

【図１９】本発明の第７の実施の形態を示す図である。

【図２０】本発明の第８の実施の形態を示す図である。

【図２１】本発明の第９の実施の形態を示す図である。

【符号の説明】

Ｔ／Ｃ・・・タイミング／コントロール回路

Ｓｍｅｍ、Ｎ’ｍｅｍ、Ｎｍｅｍ、Ｍｍｅｍ、Ａ’ｍｅｍ、Ｗｌｍｅｍ、Ｗｈｍｅｍ・・・演算値メモリ

Ｍｕｌ／Ａｄｄ・・・高速乗加算器

Ａｄｄ・・・高速加算器

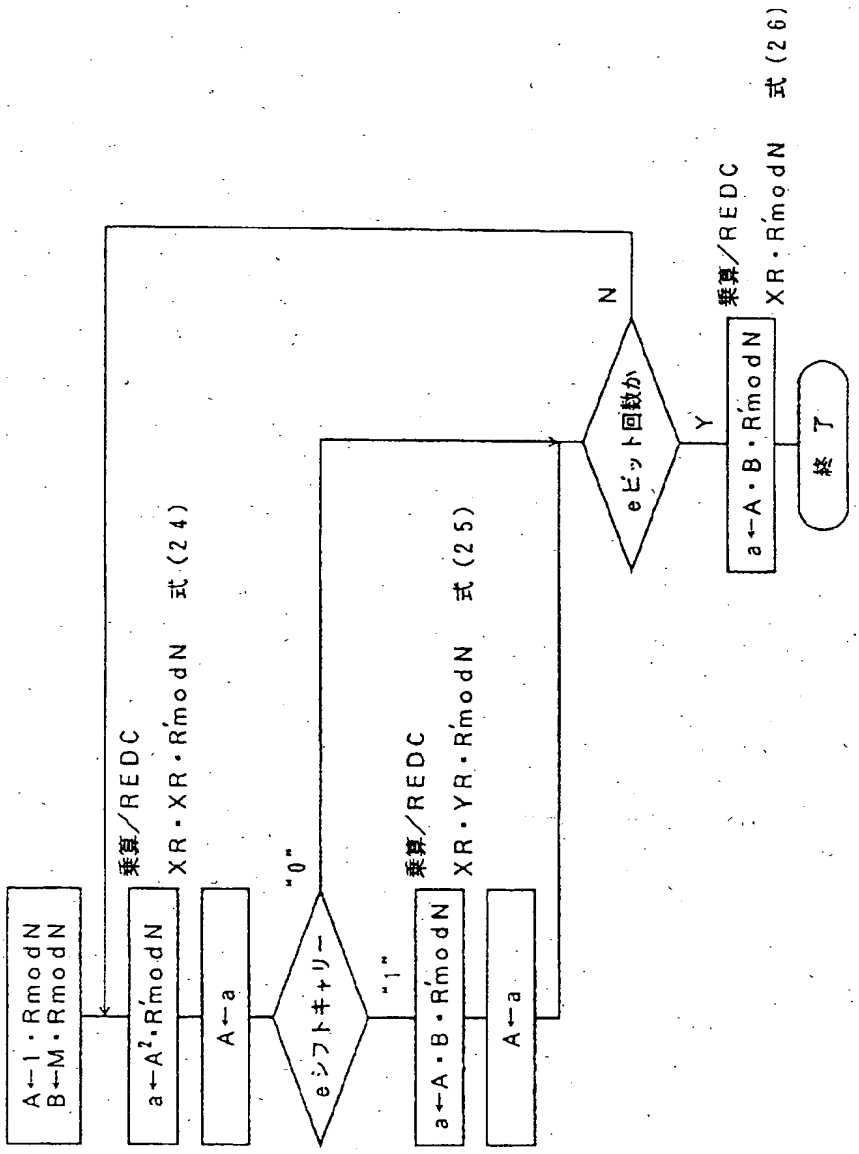
41

Ｘｉ－ｒｅｇ・・・乗算値格納レジスタ
Ｙｉ－ｒｅｇ・・・被乗算値格納レジスタ
Ａｉ－ｒｅｇ・・・被加算値格納レジスタ

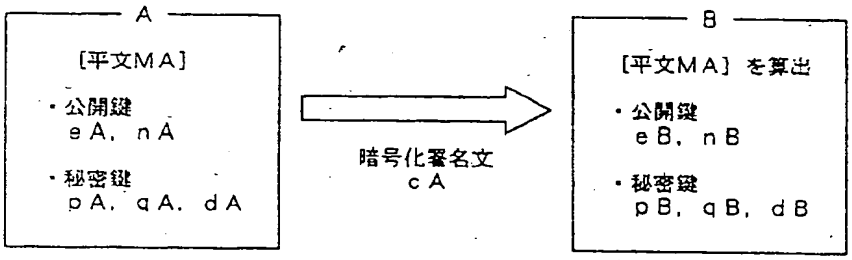
42

ＲＨ・・・高速加算器Ａｄｄが出力する値の上位の桁を格納するレジスタ

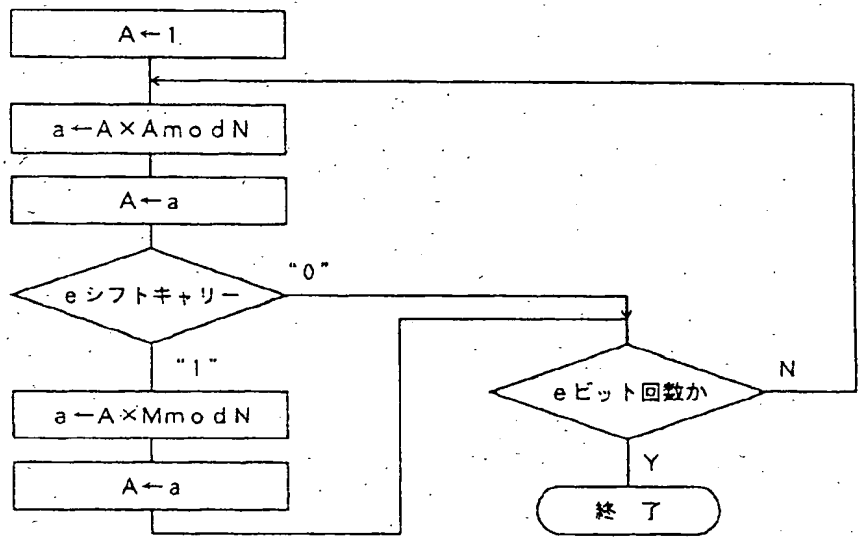
【図１】



【図2】

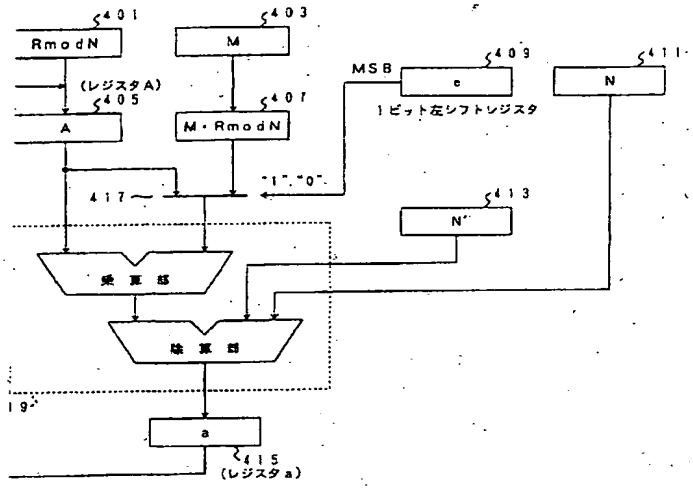


【図3】

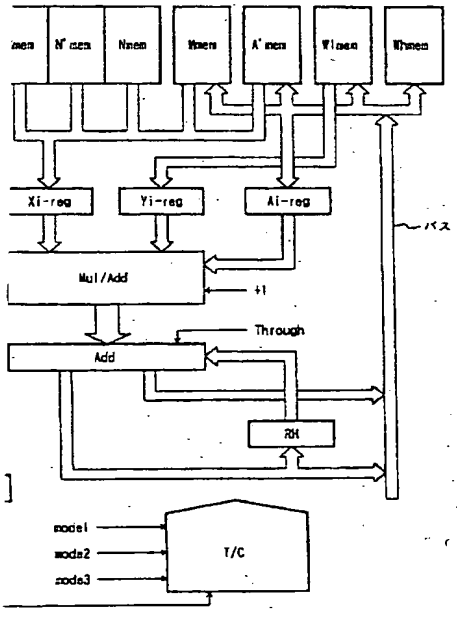


べき乗剰余演算（M^e modN）概略フロー

【図4】



【図5】



第1の実施の形態

【図6】

乗算入力処理ビット長（4ビット）

A × A = A'3 A'2 A'1 A'0 × W13 W12 W11 W10

演算ビット長（16ビット）

A'3 A'2 A'1 A'0 ←― A'memの値

×) W13 W12 W11 W10 ←― W1memの値

A'0×W10 ←― 単位乗加算

A'1×W10

A'2×W10

A'3×W10

A'0×W11

A'1×W11

【図7】

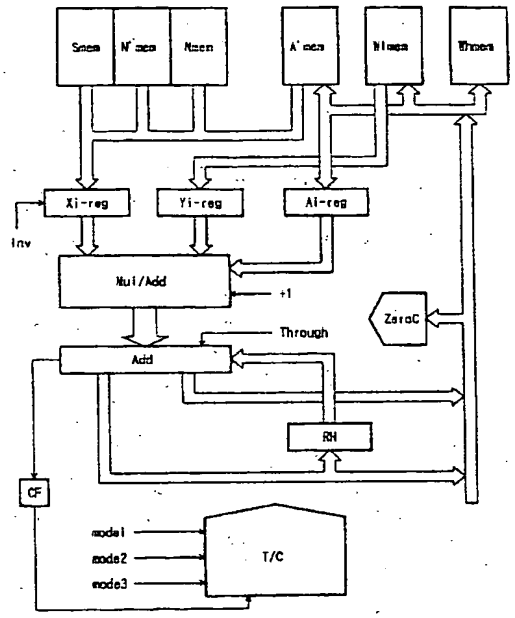
A'3 A'2 A'1 A'0 × W13 W12 W11 W10 = Wh3 Wh2 Wh1 Wh0 W13 W12 W11 W10

① A'0 × W10 + 0 = RH W10
② A'1 × W10 + 0 + RH = RH Wh0
③ A'2 × W10 + 0 + RH = RH Wh1
④ A'3 × W10 + 0 + RH = Wh3 Wh2
⑤ A'0 × W11 + Wh0 = RH W11
⑥ A'1 × W11 + Wh1 + RH = RH Wh0
⑦ A'2 × W11 + Wh2 + RH = RH Wh1
⑧ A'3 × W11 + Wh3 + RH = Wh3 Wh2
⑨ A'0 × W12 + Wh0 = RH W12
⑩ A'1 × W12 + Wh1 + RH = RH Wh0
⑪ A'2 × W12 + Wh2 + RH = RH Wh1
⑫ A'3 × W12 + Wh3 + RH = Wh3 Wh2
⑬ A'0 × W13 + Wh0 = RH W13
⑭ A'1 × W13 + Wh1 + RH = RH Wh0
⑮ A'2 × W13 + Wh2 + RH = RH Wh1
⑯ A'3 × W13 + Wh3 + RH = Wh3 Wh2

[Whmem-Wlmem ＊]

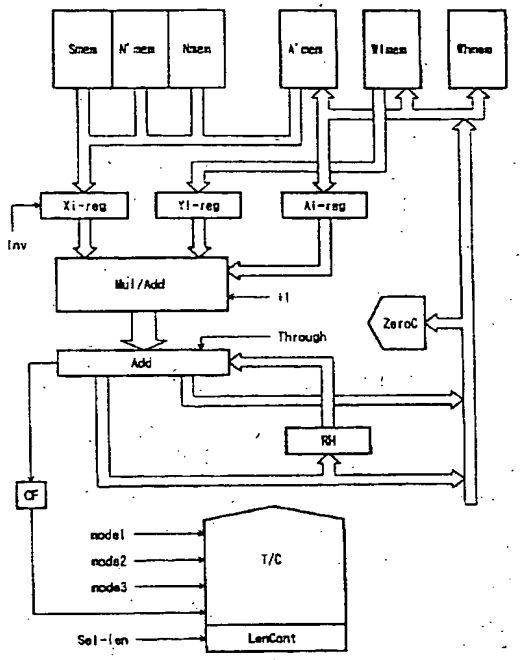
例えば 8ビット × 8ビット の計算 6AH × F5H = 6572H の場合の計算結果 6572H について、
65H を Whmem に、72H を Wlmem にそれぞれ格納することを示す。

【図８】



第２の実施の形態

【図９】



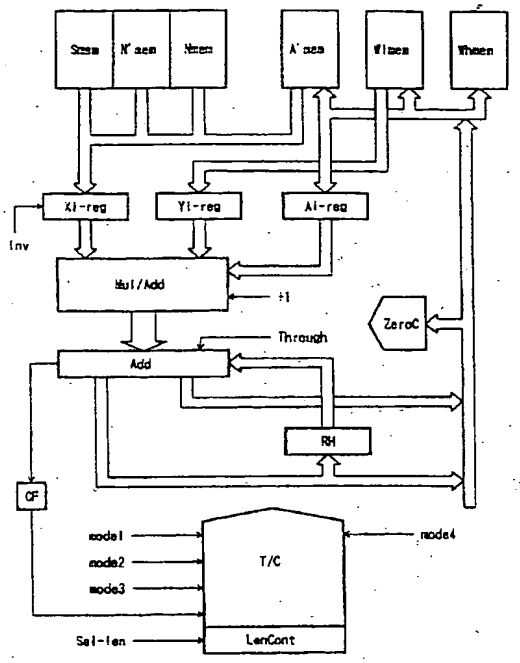
第３の実施の形態

【図１１】

S3 S2 S1 S0 × W13 W12 W11 W10 ＋ Wh3 Wh2 Wh1 Wh0 ＝ Wh3 Wh2 Wh1 Wh0 W13 W12 W11 W10

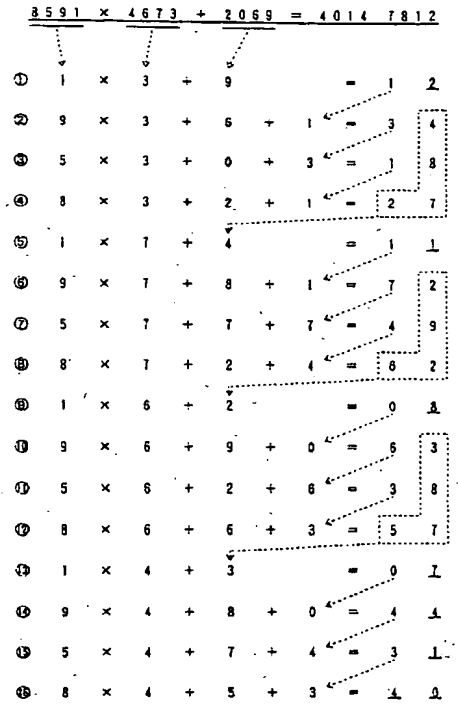
① S0 × W10 ＋ Wh0 ＝ RH W10
② S1 × W10 ＋ Wh1 ＋ RH ＝ RH Wh0
③ S2 × W10 ＋ Wh2 ＋ RH ＝ RH Wh1
④ S3 × W10 ＋ Wh3 ＋ RH ＝ Wh3 Wh2
⑤ S0 × W11 ＋ Wh0 ＝ RH W11
⑥ S1 × W11 ＋ Wh1 ＋ RH ＝ RH Wh0
⑦ S2 × W11 ＋ Wh2 ＋ RH ＝ RH Wh1
⑧ S3 × W11 ＋ Wh3 ＋ RH ＝ Wh3 Wh2
⑨ S0 × W12 ＋ Wh0 ＝ RH W12
⑩ S1 × W12 ＋ Wh1 ＋ RH ＝ RH Wh0
⑪ S2 × W12 ＋ Wh2 ＋ RH ＝ RH Wh1
⑫ S3 × W12 ＋ Wh3 ＋ RH ＝ Wh3 Wh2
⑬ S0 × W13 ＋ Wh0 ＝ RH W13
⑭ S1 × W13 ＋ Wh1 ＋ RH ＝ RH Wh0
⑮ S2 × W13 ＋ Wh2 ＋ RH ＝ RH Wh1
⑯ S3 × W13 ＋ Wh3 ＋ RH ＝ Wh3 Wh2

【図１０】

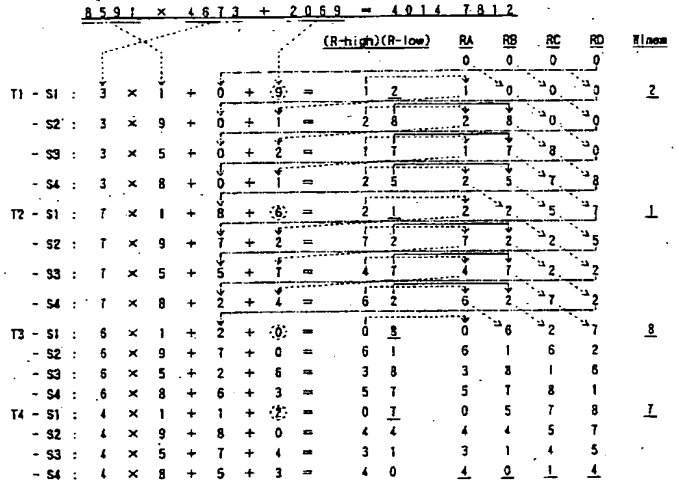


第４の実施の形態

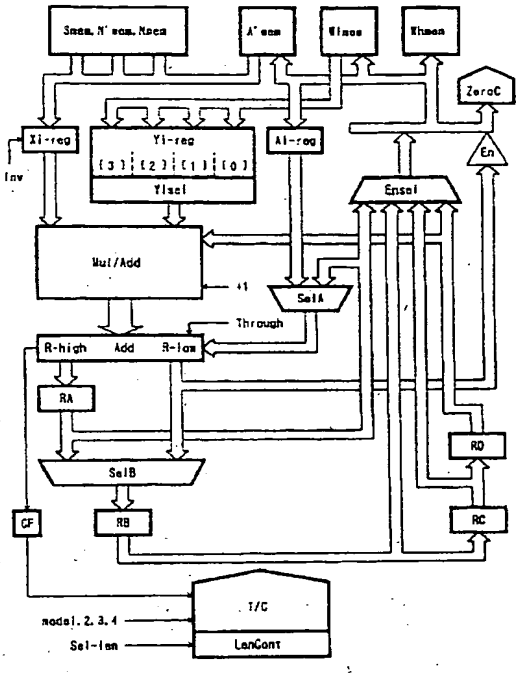
【図１２】



【図１７】

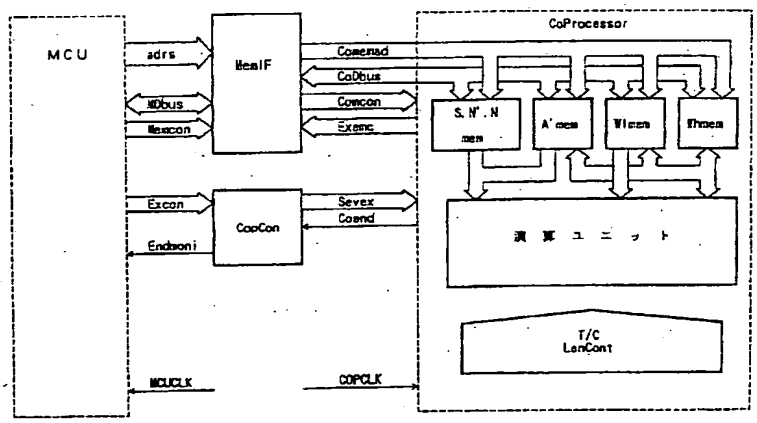


【図１３】



第５の実施の形態

【図１８】



第６の実施の形態

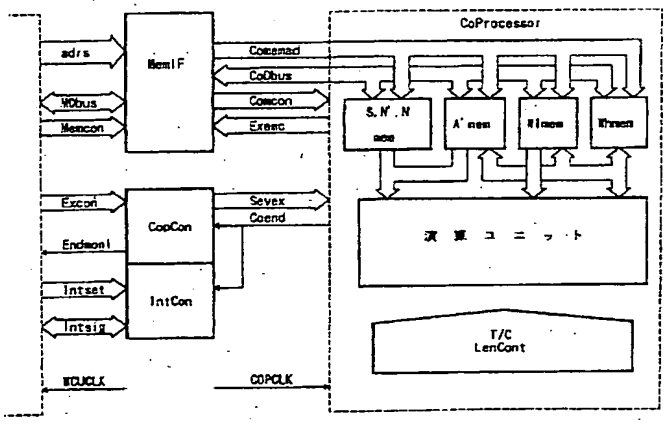
【図１４】

A3 A2 A1 A0 · B0 + C0

W13 W12 W11 W10 × S0 + Wh0 = RA RB RC RD W10

設定]
\ = RB = RC = RD = "0" . Xi-reg ← S0 , Ai-reg ← Wh0
i-reg[0] ← W10 , Yi-reg[1] ← W11 , Yi-reg[2] ← W12 , Yi-reg[3] ← W13
[Step1]
Xi-reg × Yi-reg[0] + RD + Ai-reg = (R-high)(R-low)
RD ← RC , RC ← RB , RB ← RA
RA ← (R-high) , W10← (R-low)
[Step2]
Xi-reg × Yi-reg[1] + RD + RA = (R-high)(R-low)
RD ← RC , RC ← RB
RA ← (R-high) , RB ← (R-low)
[Step3]
Xi-reg × Yi-reg[2] + RD + RA = (R-high)(R-low)
RD ← RC , RC ← RB
RA ← (R-high) , RB ← (R-low)
[Step4]
Xi-reg × Yi-reg[3] + RD + RA = (R-high)(R-low)
RD ← RC , RC ← RB
RA ← (R-high) , RB ← (R-low)

【図１９】



第７の実施の形態

【図１５】

A3 A2 A1 A0 ・ B3 B2 B1 B0 ＋ C3 C2 C1 C0

W13 W12 W11 W10 × S3 S2 S1 S0 ＋ Wh3 Wh2 Wh1 Wh0 ＝ RA RB RC RD W13 W12 W11 W10

[初期設定]
RA ＝ RB ＝ RC ＝ RD ＝ "0" ， Xi-reg ← S0 ， Ai-reg ← Wh0
Yi-reg[0] ← W10 ， Yi-reg[1] ← W11 ， Yi-reg[2] ← W12 ， Yi-reg[3] ← W13

<Time1>

[Step1]
Xi-reg × Yi-reg[0] ＋ RD ＋ Ai-reg ＝ (R-high)(R-low)
RD ← RC ， RC ← RB ， RB ← RA ， RA ← (R-high) ， W10 ← (R-low)

[Step2]
Xi-reg × Yi-reg[1] ＋ RD ＋ RA ＝ (R-high)(R-low)
RD ← RC ， RC ← RB ， RA ← (R-high) ， RB ← (R-low)

[Step3]
Xi-reg × Yi-reg[2] ＋ RD ＋ RA ＝ (R-high)(R-low)
RD ← RC ， RC ← RB ， RA ← (R-high) ， RB ← (R-low)

[Step4]
Xi-reg × Yi-reg[3] ＋ RD ＋ RA ＝ (R-high)(R-low)
RD ← RC ， RC ← RB ， RA ← (R-high) ， RB ← (R-low)

＊ Step2 から Step4 までの期間に、各メモリのアドレスの変更、プリチャージ、Xi-reg へのデータ格納、Ai-reg へのデータ格納をそれぞれ行う。

Xi-reg ← S1 ， Ai-reg ←Wh1

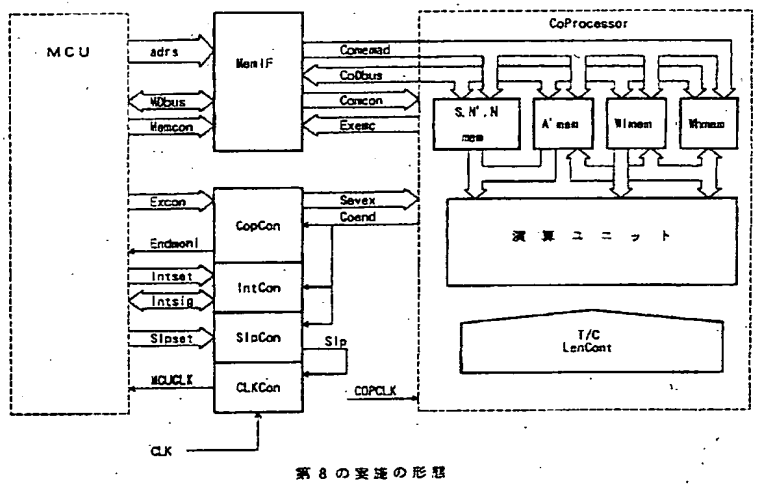
【図１６】

<Time2>
　[Step1]
　　W11 ← (R-low)　の操作以外は Time1 の場合と同様。
　[Step2] ～ [Step4]
　　Time1 の場合とまったく同様。
　　Xi-reg ← S2 ,　Ai-reg ←Wh2

<Time3>
　[Step1]
　　W12 ← (R-low)　の操作以外は Time1 の場合と同様。
　[Step2] ～ [Step4]
　　Time1 の場合とまったく同様。
　　Xi-reg ← S3 ,　Ai-reg ←Wh3

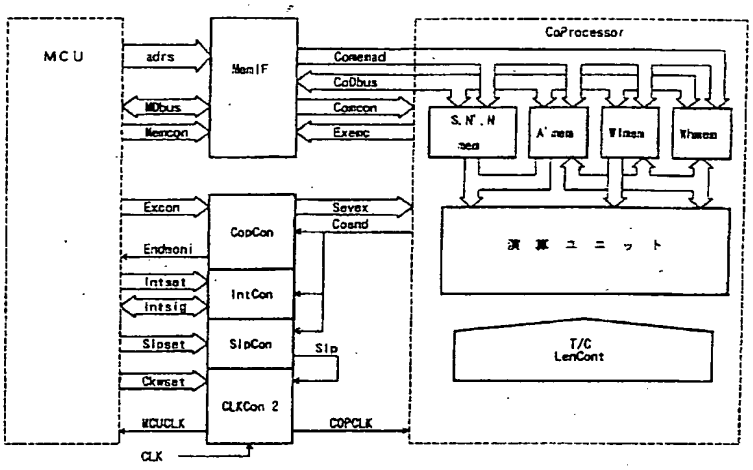
<Time4>
　[Step1]
　　W13 ← (R-low)　の操作以外は Time1 の場合と同様。
　[Step2] ～ [Step4]
　　Time1 の場合とまったく同様。

【図２０】



第８の実施の形態

【図２１】



第９の実施の形態

フロントページの続き

(72)発明者　川▲崎▼　清人
宮崎県児湯郡高鍋町大字南高鍋569番地３
合資会社川▲崎▼電機内